



TCAM/TwinCADtm

Command Language Reference Manual

V 3.1

Author: Kang, Hsin Min

Copyright © 1993-1999 By TCAM Development House Co., Ltd., Taiwan, R.O.C.

TCAM is a registered trademark of TCAM Development House Co., Ltd.

TCAM/TurboCAD is a registered trademark of TCAM Development House Co., Ltd.

TCAM/TwinCAD is a trademark of TCAM Development House Co., Ltd.

TCL is a trademark of TCAM Development House Co., Ltd.

Revision Date: November, 1999

TCAM/TwinCAD™ Command Language Reference Manual V3.1

Copyright© 1993-1999, By Kang, Hsin Min & TCAM Development House Co., Ltd.

All rights reserved. No part of this publication may be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording, or by any other information storage and retrieval system, without permission in writing from TCAM Development House Co., Ltd.

TCAM Development House Co., Ltd.

8F, No. 12, Lane 83, Sec.1, Guang Fuh Rd., San-Chung, Taipei Hsien, Taiwan, R.O.C.

Tel: 886-2-29990460

Fax: 886-2-29990461

Web: <http://www.tcaml.com.tw>

Email: support@twincad.com

This material and the products it describes are provided in an “AS-IS” basis. TCAM Development House Co., Ltd. makes no warranty, either expressed or implied, regarding it. In no event shall TCAM Development House Co., Ltd. be liable to anyone for any special, incidental or consequential damages in connection with or arising out of the purchase or use of this material.

TCAM Development House Co., Ltd. reserves the rights to revise and improve its products as it sees fit. This publication describes the state of this product at the time of its publication, and may not reflect the product at all times in the future.

Author: Kang, Hsin-Min

Trademark Acknowledgements

TCAM is a registered trademark of TCAM Development House Co., Ltd.

TCAM/TurboCAD is a registered trademark of TCAM Development House Co., Ltd.

TCAM/TwinCAD is a trademark of TCAM Development House Co., Ltd.

TCL is a trademark of TCAM Development House Co., Ltd.

IBM is a registered trademark of International Business Machine Corp.

Windows is a registered trademark of Microsoft Corporation.

AutoCAD is a registered trademark of Autodesk, Inc.

All other brand and product names are trademarks or registered trademarks of their respective holders.

Table of Contents

PART 1

THE TWINCAD COMMAND LANGUAGE V3.1

OVERVIEW	REF - 2
Introduction to TCL.....	REF - 2
Usefulness of TCL to TwinCAD	REF - 2
Extending TwinCAD with TCL.....	REF - 3
Writing TCL Programs Using Text Editor	REF - 4
Protecting Your TCL Programs	REF - 5
The TCL.EXE Utility.....	REF - 5
The TCZ file.....	REF - 6
Binding Application With TwinCAD.....	REF - 6
TCL LEXICAL ELEMENTS.....	REF - 8
The Source File And Character Set	REF - 8
Line Break and White Space	REF - 8
Comments.....	REF - 8
Token.....	REF - 8
Operators and separators	REF - 9
Identifiers	REF - 9
Predefined Identifiers	REF - 9
Embedded Identifiers	REF - 10
Reserved words	REF - 10
Constants	REF - 10
Integer constants.....	REF - 11
Floating point constants	REF - 11
Character Constant	REF - 11
Character Escape Codes	REF - 11
String constants	REF - 12
TCL Registers	REF - 12
Special Note on Backslash Code in String Constant	REF - 13
THE TCL PREPROCESSOR.....	REF - 14
#define	REF - 14
Predefined Macro	REF - 16
#include	REF - 16
#ifdef and #ifndef	REF - 17
#else and #endif.....	REF - 17
#ifyes	REF - 17
#outmsg	REF - 18
#codechk.....	REF - 18
Examples.....	REF - 18
DECLARATIONS	REF - 20
Variable Declarations	REF - 20
Scope of Declaration	REF - 20
Data Type and Type Specifier	REF - 21
Initializer	REF - 22
Array Variable Declarations	REF - 23
Function Declarations.....	REF - 23
Class of Function.....	REF - 23
Type of Function.....	REF - 24
Parameter List in Function	REF - 24
Predefined Typedef Data Types.....	REF - 26
POINT Data Type	REF - 26
LINE Data Type.....	REF - 26
ARC Data Type.....	REF - 26
ELLIPSE Data Type.....	REF - 26

CIRCLE Data Type	REF - 27
LAYER Data Type	REF - 27
LTYPE Data Type	REF - 27
STYLE Data Type	REF - 28
FILE Data Type	REF - 28
ENTITY Data Type	REF - 28
SLIST Data Type	REF - 28
ENTDATA Data Type	REF - 29
TCL EXPRESSIONS	REF - 39
L-value	REF - 39
Precedence and Associativity	REF - 39
List of Operators	REF - 40
Difference in Operators	REF - 41
Comma Expressions	REF - 42
Comma Expressions for Assignment	REF - 44
Scalar Data Conversions	REF - 44
POINT Data Conversions	REF - 44
LINE Data Conversions	REF - 45
ARC Data Conversions	REF - 45
CIRCLE Data Conversions	REF - 46
ELLIPSE Data Conversions	REF - 46
TCL CONTROL FLOW	REF - 48
Statements	REF - 48
Conditional Statements	REF - 48
Switch Statement	REF - 49
Iterative Statements	REF - 50
while Statement	REF - 50
do Statement	REF - 50
For Statement	REF - 50
Break Statement	REF - 51
Continue Statement	REF - 51
Return Statement	REF - 51
TCL PROGRAM STRUCTURE	REF - 52
The Main Program	REF - 52
CLOAD and RUN	REF - 52
COMMAND Class	REF - 53
FUNCTION Class	REF - 53

PART 2

GENERAL OF TWINCAD SYSTEM VARIABLES

@DISPMODE	VAR - 1
@VIEWX	VAR - 2
@VIEWY	VAR - 2
@VIEWZ	VAR - 2
@ACADTADJ	VAR - 2
@AJOINEPS	VAR - 3
@ALLTRNSCMD	VAR - 3
@ANSWERMODE	VAR - 3
@APERTURE	VAR - 3
@ARCSEGANG	VAR - 3
@ASKFORTAG	VAR - 3
@AUTOAUDIT	VAR - 4
@AUTOQTEXT	VAR - 4
@AUTOREDRAW	VAR - 4
@AUTOREGION	VAR - 5
@AUTOSAVE	VAR - 5
@AXOSCALE	VAR - 5
@BEVANGLE	VAR - 6
@BLIPMODE	VAR - 6
@BLOCKCTRL	VAR - 6
@BPOLYLEVEL	VAR - 7
@BPOLYMODE	VAR - 7
@BTRIMMODE	VAR - 7
@BYBLKCOLOR	VAR - 8

@CDSPFILE Dos.....	VAR - 8
@CHAMDIST1.....	VAR - 8
@CHAMDIST2.....	VAR - 8
@CHELPFILE.....	VAR - 8
@CMDECHO.....	VAR - 8
@CMENUEFILE.....	VAR - 9
@COLORMASK.....	VAR - 9
@CSYMLIB.....	VAR - 9
@CURCOLOR.....	VAR - 9
@CURLAYER.....	VAR - 9
@CURLTYPE.....	VAR - 9
@CWORKFILE.....	VAR - 10
@DIMADPRE.....	VAR - 10
@DIMAOPT.....	VAR - 10
@DIMARND.....	VAR - 11
@DIMASIZE.....	VAR - 11
@DIMASTR.....	VAR - 11
@DIMATYPE.....	VAR - 11
@DIMCEN.....	VAR - 11
@DIMCOLOR.....	VAR - 11
@DIMDBSTR.....	VAR - 12
@DIMDESTR.....	VAR - 12
@DIMDLIR.....	VAR - 12
@DIMDOPT.....	VAR - 12
@DIMDPRE.....	VAR - 13
@DIMDRLEAD.....	VAR - 13
@DIMESTYLE.....	VAR - 14
@DIMEXE.....	VAR - 14
@DIMEXO.....	VAR - 14
@DIMGAP.....	VAR - 14
@DIMLFAC.....	VAR - 14
@DIMLOPT.....	VAR - 14
@DIMLTOL.....	VAR - 15
@DIMOBLANG.....	VAR - 16
@DIMOBLARW.....	VAR - 16
@DIMPFIX.....	VAR - 16
@DIMPOST.....	VAR - 16
@DIMRBSTR.....	VAR - 16
@DIMRESTR.....	VAR - 16
@DIMRND.....	VAR - 16
@DIMROPT.....	VAR - 17
@DIMSCALE.....	VAR - 18
@DIMSE1.....	VAR - 18
@DIMSE2.....	VAR - 18
@DIMITADJ.....	VAR - 18
@DIMITCDIST.....	VAR - 18
@DIMITFAC.....	VAR - 19
@DIMITOL.....	VAR - 19
@DIMTRDIST.....	VAR - 19
@DIMTSIZE.....	VAR - 19
@DIMITSTYLE.....	VAR - 20
@DIMITWIDTH.....	VAR - 20
@DIMUTOL.....	VAR - 20
@DISPCTRL.....	VAR - 20
@DISP_POSNO.....	VAR - 20
@DO3DMODE.....	VAR - 21
@DWGCOLOR.....	VAR - 21
@DWGINPURGE.....	VAR - 21
@DWGVERSION.....	VAR - 22
@DXFIOCNV.....	VAR - 22
@ECS_NX.....	VAR - 22
@ECS_NY.....	VAR - 22
@ECS_NZ.....	VAR - 22
@ECS_ORG.....	VAR - 23
@ECS_ORGZ.....	VAR - 23
@ECS_XDIR.....	VAR - 23
@ECS_XDIRZ.....	VAR - 23

@ECS_ZH.....	VAR - 23
@EIDLEVEL.....	VAR - 23
@ELEVATION.....	VAR - 23
@ELLIPSEARC.....	VAR - 24
@ENCRYPT.....	VAR - 24
@ENTORDER.....	VAR - 24
@EXPERT.....	VAR - 24
@EXPLOLIMIT.....	VAR - 25
@EXPLOTEXT.....	VAR - 25
@EXTDEXACT.....	VAR - 25
@EXTFNTYPE.....	VAR - 25
@FASTLTYPE.....	VAR - 26
@FILLETOPT.....	VAR - 26
@FILLETRAD.....	VAR - 26
@FILLMODE.....	VAR - 26
@FONTOPTION.....	VAR - 26
@GRIDCOLOR.....	VAR - 27
@GRIDUNIT.....	VAR - 27
@HIDEANONYM.....	VAR - 27
@HIGHLIGHT.....	VAR - 27
@INPSTATE Dos.....	VAR - 28
@INSBASE.....	VAR - 28
@INSDMODE.....	VAR - 28
@LIMMAX.....	VAR - 29
@LIMMIN.....	VAR - 29
@LTMASK.....	VAR - 29
@LTSCALE.....	VAR - 29
@MAXCHANGEPT.....	VAR - 29
@MAXPVANG.....	VAR - 29
@MAXPVLEN.....	VAR - 30
@MAXRECLINE.....	VAR - 30
@MENUINUSE Dos.....	VAR - 30
@MENUTYPE Dos.....	VAR - 30
@MINDONUT.....	VAR - 31
@MIRRTEXT.....	VAR - 31
@MVSCALE.....	VAR - 31
@NOEMPTYBLK.....	VAR - 32
@OSMODE.....	VAR - 32
@PDMODE.....	VAR - 32
@PDSIZE.....	VAR - 33
@PENCOLOR.....	VAR - 33
@PICKBOX.....	VAR - 33
@PICKQUERY.....	VAR - 33
@PINPFILE Dos.....	VAR - 33
@PLAYDELAY Dos.....	VAR - 34
@PLOTOPTION.....	VAR - 34
@POFSTCANG.....	VAR - 34
@PPLSYMBOL.....	VAR - 35
@PREVIEWOPT.....	VAR - 35
@QTEXTGENR.....	VAR - 35
@QTEXTMODE.....	VAR - 36
@RECSTATE.....	VAR - 36
@RINPFILE Dos.....	VAR - 36
@SARCTYPE.....	VAR - 36
@SAVEBACKUP.....	VAR - 38
@SAVEFILE.....	VAR - 38
@SAVETIME.....	VAR - 38
@SELMASK.....	VAR - 38
@SHOWDIR.....	VAR - 39
@SNAPANGLE.....	VAR - 39
@SNAPBASE.....	VAR - 39
@SNAPBETA.....	VAR - 39
@SNAPFLAG.....	VAR - 40
@SNAPGAMMA.....	VAR - 40
@SNAPUNIT.....	VAR - 41
@SPLFRAME.....	VAR - 41
@SPLINESEG.....	VAR - 41

@SPLINETYPE.....	VAR - 41
@SVARORDER.....	VAR - 42
@SYMCOLOR.....	VAR - 42
@SYMESTYLE.....	VAR - 42
@SYMSTYLE.....	VAR - 43
@SYSENDFLAG.....	VAR - 43
@SYSOPTION Win.....	VAR - 43
@SYSPROFILE Win.....	VAR - 44
@SYSVERSION.....	VAR - 44
@TCAMDXFEXT.....	VAR - 44
@TXTLTYPE.....	VAR - 45
@TGFTYPE.....	VAR - 45
@TG_FILE.....	VAR - 45
@THICKNESS.....	VAR - 45
@TXTBOXGAP.....	VAR - 45
@TXTCSPACE.....	VAR - 46
@TXTESTYLE.....	VAR - 46
@TXTHEIGHT.....	VAR - 46
@TXTOBANGLE.....	VAR - 46
@TXTOBX.....	VAR - 46
@TXTROTCCW.....	VAR - 47
@TXTROWDIST.....	VAR - 47
@TXTSTYLE.....	VAR - 47
@TXTSUBFAC.....	VAR - 47
@TXTWIDTHF.....	VAR - 48
@UCSCONTROL.....	VAR - 48
@UCSORG.....	VAR - 48
@UCSORGZ.....	VAR - 48
@UCSSIZE.....	VAR - 48
@UCSSYMBOL.....	VAR - 49
@UNDOCTRL.....	VAR - 49
@UNITALPHA.....	VAR - 49
@UNITBETA.....	VAR - 49
@UNITGAMMA.....	VAR - 49
@USEAUXMENU Dos.....	VAR - 50
@USERVARINF.....	VAR - 50
@USE_EMODE.....	VAR - 50
@VWNDMAX.....	VAR - 51
@VWNDMIN.....	VAR - 51
@VZCENTER.....	VAR - 51
@VZSCALE.....	VAR - 51
@WIDAUXMENU Dos.....	VAR - 51
@WIDCMDAREA Dos.....	VAR - 51
@WIDRAWING Dos.....	VAR - 51
@WIDSCRMENU Dos.....	VAR - 51
@WIDSTATUS Dos.....	VAR - 52
@WORKEMAX.....	VAR - 52
@WORKEMIN.....	VAR - 52
@WORKSTATUS.....	VAR - 52
@ZESCALE.....	VAR - 52
@ZTESTEPS.....	VAR - 52

PART 3

TCL RUNTIME LIBRARY V3.10

OVERVIEW OF TCL RUNTIME LIBRARY V3.10.....	FNC - 1
COMPATIBILITY BETWEEN WINDOWS AND DOS VERSION.....	FNC - 3
a of A.....	FNC - 4
abs.....	FNC - 5
acos.....	FNC - 5
acosr.....	FNC - 6
add.....	FNC - 6
add_sym.....	FNC - 6
add_tag.....	FNC - 8
asin.....	FNC - 9

asinr	FNC - 9
atan	FNC - 10
atan2	FNC - 10
atan2r	FNC - 10
atanr	FNC - 11
beep	FNC - 11
block	FNC - 11
c or C	FNC - 12
c_pccs	FNC - 13
c_plcs	FNC - 13
c_plls	FNC - 13
c_ppa	FNC - 14
c_ppcs	FNC - 14
c_ppls	FNC - 14
c_ppp	FNC - 15
c_ppr	FNC - 15
card_mid	FNC - 15
card_zid	FNC - 16
cb_dump Win	FNC - 16
cb_empty Win	FNC - 16
cb_format Win	FNC - 16
cb_getdata Win	FNC - 17
cb_load Win	FNC - 18
cb_query Win	FNC - 18
cb_setdata Win	FNC - 19
cc_pos	FNC - 20
ceil	FNC - 20
centroid	FNC - 20
chmod	FNC - 21
clayer	FNC - 22
clip_arc	FNC - 22
clip_line	FNC - 23
codetype Win	FNC - 23
com_close	FNC - 24
com_getc	FNC - 24
com_open	FNC - 24
com_putc	FNC - 26
com_state	FNC - 27
com_ugetc	FNC - 28
command	FNC - 28
copyfile	FNC - 33
cos	FNC - 33
cosr	FNC - 33
crc_check	FNC - 34
cspline	FNC - 34
date	FNC - 35
date\$	FNC - 35
db_append	FNC - 36
db_close	FNC - 36
db_create	FNC - 36
db_delete	FNC - 37
db_deleted	FNC - 37
db_fid	FNC - 38
db_field	FNC - 38
db_fieldno	FNC - 38
db_get	FNC - 38
db_goto	FNC - 39
db_insert	FNC - 39
db_locate	FNC - 39
db_logic	FNC - 40
db_pack	FNC - 40
db_put	FNC - 41
db_recall	FNC - 41
db_reccount	FNC - 41
db_recno	FNC - 41
db_select	FNC - 42
db_skip	FNC - 42

db_sort	FNC - 42
db_use	FNC - 43
db_value	FNC - 43
db_zap	FNC - 44
dbl	FNC - 44
dde_close Win	FNC - 44
dde_closeall Win	FNC - 44
dde_coldlink Win	FNC - 45
dde_error Win	FNC - 45
dde_execute Win	FNC - 45
dde_hotlink Win	FNC - 46
dde_get Win	FNC - 46
dde_getline Win	FNC - 47
dde_offset Win	FNC - 47
dde_open Win	FNC - 48
dde_poke Win	FNC - 48
dde_request Win	FNC - 49
dde_state Win	FNC - 50
dde_timeout Win	FNC - 51
dde_warmlink Win	FNC - 51
delay	FNC - 51
DeviceCaps Win	FNC - 52
dgt_btn	FNC - 53
dgt_load	FNC - 53
dgt_off	FNC - 53
dgt_on	FNC - 53
dgt_pos	FNC - 54
dgt_save	FNC - 54
dgt_state	FNC - 54
dim_text	FNC - 54
diskfree	FNC - 55
div	FNC - 55
dtr	FNC - 56
e or E	FNC - 56
eblock	FNC - 56
ecs_org	FNC - 57
edrvtype	FNC - 58
ee_slineFNC -	FNC - 58
eid	FNC - 59
ent_alloc	FNC - 59
ent_area	FNC - 60
ent_bnext	FNC - 60
ent_break	FNC - 61
ent_copy	FNC - 61
ent_count	FNC - 62
ent_erase	FNC - 62
ent_init	FNC - 63
ent_intsc	FNC - 66
ent_last	FNC - 66
ent_len	FNC - 66
ent_main	FNC - 67
ent_next	FNC - 67
ent_null	FNC - 67
ent_ofst	FNC - 67
ent_ok	FNC - 68
ent_plast	FNC - 69
ent_pnext	FNC - 69
ent_pseg	FNC - 69
ent_read	FNC - 70
ent_rvs	FNC - 70
ent_state	FNC - 70
ent_tnext	FNC - 71
ent_trns	FNC - 71
ent_type	FNC - 72
ent_write	FNC - 72
eto_wcs	FNC - 73
eval	FNC - 73

exbuf_io.....	FNC - 73
exec.....	FNC - 74
exit.....	FNC - 76
exp.....	FNC - 76
fclose.....	FNC - 76
fcloseall.....	FNC - 76
feof.....	FNC - 77
fexist.....	FNC - 77
fget.....	FNC - 77
fgets.....	FNC - 78
filesize.....	FNC - 78
FindWindow Win.....	FNC - 79
fit_circle.....	FNC - 79
fit_line.....	FNC - 79
fix.....	FNC - 80
floor.....	FNC - 80
fopen.....	FNC - 80
fprintf.....	FNC - 80
fput.....	FNC - 81
fputs.....	FNC - 81
frac.....	FNC - 81
fsearch.....	FNC - 82
fseek.....	FNC - 82
fstrloc.....	FNC - 82
ftell.....	FNC - 83
fullname.....	FNC - 83
gauss.....	FNC - 83
gcd.....	FNC - 84
get_dirlist.....	FNC - 84
get_fontinf.....	FNC - 85
get_profile Win.....	FNC - 86
get_syspath.....	FNC - 87
getangle.....	FNC - 87
getblock.....	FNC - 88
getcd.....	FNC - 88
getcolor.....	FNC - 88
getcorner.....	FNC - 88
getdist.....	FNC - 89
gete.....	FNC - 89
getenv.....	FNC - 91
getesel.....	FNC - 91
getfile.....	FNC - 94
getfsize.....	FNC - 95
getinput.....	FNC - 96
getkey.....	FNC - 96
getkeyword.....	FNC - 97
getlayer.....	FNC - 98
getln.....	FNC - 98
GetLongName.....	FNC - 99
getltype.....	FNC - 99
getp.....	FNC - 100
getpath.....	FNC - 100
GetRGBColor Win.....	FNC - 100
gets.....	FNC - 100
GetShortName.....	FNC - 101
getstyle.....	FNC - 101
getv.....	FNC - 101
gextent.....	FNC - 102
gr_screen DOS.....	FNC - 103
GUIColor Win.....	FNC - 103
GUIFont Win.....	FNC - 104
i or I.....	FNC - 106
identify.....	FNC - 106
ifix.....	FNC - 107
in_span.....	FNC - 107
index.....	FNC - 107
inp.....	FNC - 108

inpw.....	FNC - 108
is_running Win.....	FNC - 108
kbdstate.....	FNC - 108
keyin.....	FNC - 109
l or L.....	FNC - 109
l_cctan.....	FNC - 110
layer.....	FNC - 111
layer_off.....	FNC - 112
layer_on.....	FNC - 112
layerid.....	FNC - 112
left\$.....	FNC - 113
len.....	FNC - 113
linetype.....	FNC - 113
ln.....	FNC - 114
lower.....	FNC - 114
ltypeid.....	FNC - 115
log.....	FNC - 115
max.....	FNC - 115
memmove.....	FNC - 115
memset.....	FNC - 116
menucmd.....	FNC - 116
MessageBox Win.....	FNC - 117
min.....	FNC - 118
mkdir.....	FNC - 118
mod.....	FNC - 118
mul.....	FNC - 119
mv_trns.....	FNC - 119
newent.....	FNC - 119
os_language Win.....	FNC - 120
os_type.....	FNC - 121
outp.....	FNC - 122
outpw.....	FNC - 122
p or P.....	FNC - 122
p_pctan.....	FNC - 123
pe_angle.....	FNC - 124
pe_dist.....	FNC - 124
PenColor Win.....	FNC - 125
pi.....	FNC - 125
plotent.....	FNC - 125
pm_command Win.....	FNC - 126
pm_request Win.....	FNC - 126
polyeval.....	FNC - 127
polyflag.....	FNC - 128
PostMessage Win.....	FNC - 128
printf.....	FNC - 128
prints.....	FNC - 130
put_map.....	FNC - 131
put_profile Win.....	FNC - 131
pwr.....	FNC - 132
r_level.....	FNC - 133
r_simplex.....	FNC - 133
rand.....	FNC - 135
rdms.....	FNC - 135
redraw.....	FNC - 135
redraw_count Win.....	FNC - 136
regen.....	FNC - 136
regen_count Win.....	FNC - 136
remove.....	FNC - 136
rename.....	FNC - 137
rewind.....	FNC - 137
right\$.....	FNC - 137
rmdir.....	FNC - 137
rnd.....	FNC - 138
rnd2.....	FNC - 138
root2.....	FNC - 138
root3.....	FNC - 139
rtd.....	FNC - 139

run.....	FNC - 139
sarc.....	FNC - 140
search.....	FNC - 140
sel_icon.....	FNC - 141
SelectFont Win.....	FNC - 142
SendMessage Win.....	FNC - 142
series_no.....	FNC - 143
set_com.....	FNC - 143
set_drag.....	FNC - 144
set_ecs.....	FNC - 145
set_eps.....	FNC - 146
set_fkey DOS.....	FNC - 146
set_map.....	FNC - 148
set_mask.....	FNC - 151
set_mvtrns.....	FNC - 151
set_trns.....	FNC - 152
set_undo.....	FNC - 152
setdata.....	FNC - 154
setplot.....	FNC - 155
show_help.....	FNC - 156
show_icon.....	FNC - 156
sin.....	FNC - 157
sinr.....	FNC - 157
sort.....	FNC - 157
sqrt.....	FNC - 158
srand.....	FNC - 159
srch_tag.....	FNC - 159
sscanf.....	FNC - 159
strcat.....	FNC - 162
strchr.....	FNC - 162
strcmp.....	FNC - 162
strcpy.....	FNC - 163
strlen.....	FNC - 163
strmatch.....	FNC - 163
strncat.....	FNC - 164
strncmp.....	FNC - 164
strncpy.....	FNC - 165
strchr.....	FNC - 165
strev.....	FNC - 166
strrplx.....	FNC - 166
strskip.....	FNC - 166
strword.....	FNC - 167
style.....	FNC - 168
styleid.....	FNC - 169
sub.....	FNC - 169
swap.....	FNC - 169
symbol.....	FNC - 170
symlib.....	FNC - 171
sysdrive.....	FNC - 171
SystemMode Win.....	FNC - 172
sysversion.....	FNC - 172
takename.....	FNC - 173
takepath.....	FNC - 173
tan.....	FNC - 173
tanr.....	FNC - 173
tbl_count.....	FNC - 174
tbl_next.....	FNC - 174
text_len.....	FNC - 175
text_span.....	FNC - 175
tg_close.....	FNC - 175
tg_id.....	FNC - 175
tg_open.....	FNC - 176
tg_read.....	FNC - 176
tick.....	FNC - 177
time.....	FNC - 177
time\$.....	FNC - 177
timer.....	FNC - 177

ToBig5.....	FNC - 178
ToGB2312.....	FNC - 178
ToKSC.....	FNC - 179
ToolBar.....	FNC - 179
ToSJIS.....	FNC - 179
tx_close.....	FNC - 179
tx_delete.....	FNC - 180
tx_empty.....	FNC - 180
tx_insert.....	FNC - 180
tx_lineno.....	FNC - 181
tx_load.....	FNC - 181
tx_open.....	FNC - 181
tx_read.....	FNC - 181
tx_save.....	FNC - 182
tx_search.....	FNC - 182
tx_sort.....	FNC - 183
tx_write.....	FNC - 183
ufloor.....	FNC - 183
undo.....	FNC - 184
upper.....	FNC - 184
userbrk.....	FNC - 184
uservar.....	FNC - 184
v or V.....	FNC - 187
version.....	FNC - 187
wide_arc.....	FNC - 188
wide_line.....	FNC - 189
winexec Win.....	FNC - 189
wnd1btn.....	FNC - 190
wnd2btn.....	FNC - 190
wnd_bitmap Win.....	FNC - 191
wnd_button Win.....	FNC - 192
wnd_cpos Win.....	FNC - 192
wnd_editbox Win.....	FNC - 192
wnd_fgetd.....	FNC - 194
wnd_fgeto.....	FNC - 194
wnd_fgets.....	FNC - 196
wnd_field.....	FNC - 196
wnd_floct.....	FNC - 198
wnd_flow.....	FNC - 199
wnd_fmark.....	FNC - 199
wnd_ftext.....	FNC - 200
wnd_gbtn.....	FNC - 201
wnd_getsize Win.....	FNC - 202
wnd_group Win.....	FNC - 202
wnd_handle Win.....	FNC - 202
wnd_icon.....	FNC - 203
wnd_itemno.....	FNC - 204
wnd_listbox Win.....	FNC - 205
wnd_loct.....	FNC - 206
wnd_meter Win.....	FNC - 207
wnd_move Win.....	FNC - 207
wnd_puts.....	FNC - 208
wnd_rcbox Win.....	FNC - 209
wnd_rect Win.....	FNC - 210
wnd_resize Win.....	FNC - 210
wnd_rloct.....	FNC - 211
wnd_rolldn.....	FNC - 211
wnd_rollup.....	FNC - 212
wnd_runit.....	FNC - 212
wnd_select.....	FNC - 212
wnd_settext Win.....	FNC - 213
wnd_state Win.....	FNC - 214
wnd_static Win.....	FNC - 214
wnd_style Win.....	FNC - 215
wnd_textbox Win.....	FNC - 216
wnd_vport.....	FNC - 216
wnd_vscroll.....	FNC - 218

wndclose	FNC - 220
wndopen	FNC - 220
wndopensub	FNC - 222
wndtitle	FNC - 223
wto_ecs	FNC - 223
zroot2	FNC - 224
zroot3	FNC - 224
zroot4	FNC - 224

PART 4

FNC - HINTS ON TCL PROGRAMMING

Careful Use of the Command() Function	HINT - 1
Updating System Status Display	HINT - 2
Sieving Entities From a Given Selection List.....	HINT - 2
Function that Returns an SLIST data	HINT - 2
Union, Intersection and Subtraction of Selections Lists.....	HINT - 3
Modification of Polyline Segments	HINT - 3
Changing the Number of Polyline Segments.....	HINT - 3
Controlling the Joining of Polyline Segments	HINT - 4
Inserting a Specific Symbol.....	HINT - 4
Accessing the Drawing Block Defining Entities	HINT - 4
The Attribute Tags of a Block Instance.....	HINT - 5
Don't Erase Entities That Define a New Block.....	HINT - 5
Making Good Use of the Geometry Calculation Functions.....	HINT - 5
Collecting Entities Created by Command().....	HINT - 6
Use TCL Functions Instead of the DOS Command.....	HINT - 6
Shelling an External Program.....	HINT - 6
Suppressing System Messages	HINT - 7
Using the Format Specifier "%@0&f" in printf().....	HINT - 7
Passing Argument Values to an External TCL Application	HINT - 8
Returning a String from a TCL Function	HINT - 8
Determining a Unique Name for Temporary File.....	HINT - 8
Using Successive sscanf() to Parse Data String of Variable Format	HINT - 8
Sorting Multiple Index-Related Data Arrays.....	HINT - 9
The Inside-test of a Point within a Polyline.....	HINT - 10

PART 5

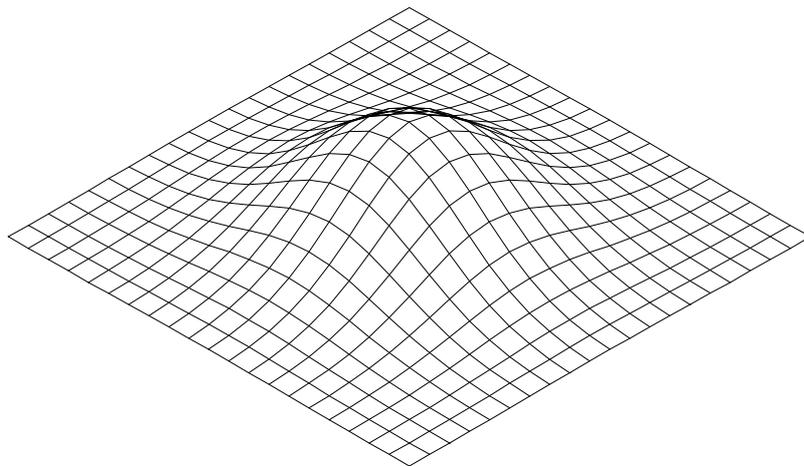
EXAMPLE TCL PROGRAMS

CDA.TCL	EX - 2
CLOCK.TCL	EX - 4
FILES.TCL	EX - 7
HANOI.TCL.....	EX - 18
LRDIM.TCL	EX - 23
LTEXT.TCL	EX - 26
NITEM.TCL	EX - 28
PISTON.TCL.....	EX - 30
PLOT2.TCL.....	EX - 33
POLYSTAR.TCL	EX - 35
SPIRAL.TCL	EX - 37
XPDL.TCL	EX - 39

APPENDIX.....	APPENDIX - 1
----------------------	---------------------

PART 1

The TwinCAD Command Language V3.1



Overview

Introduction to TCL

TCL™, short name for **TwinCAD Command Language™**, is an implementation of a C-like programming language built into the **TCAM/TwinCAD™** package. It provides the users and application developers a way to write their own programs to add commands and functions to the **TwinCAD**, in a very powerful high-level language.

TCL mimics the standard C language, with several additional features to make the programming jobs much easier for CAD/CAM applications. Among them are predefined geometry data structures, system variables, TCL geometry registers, and the related operations in expressions.

TCL is designed to have a very close relationship with the **TwinCAD** command line input. User may directly enter a TCL expression to the command line whenever a value, a point, or a string is required, as long as the expression can be evaluated to return the proper type of value. User may load in commands or functions written in TCL from a disk file, and enter the commands or reference the functions in expressions as if they were originally built into **TwinCAD**. There is no limits on the size, memory storage, and the number of TCL programs which can be loaded into the system.

To make the best use of this reference manual (sorry, not a programming tutorial), you are required to have some background in C programming. You are also encouraged to write testing programs along the path of learning. It will be a good idea for the beginners to read text books about C programming as well as the programming concepts first, before getting into TCL programming.

Usefulness of TCL to TwinCAD

The usefulness of TCL programming to **TwinCAD** can be observed from the following example.

Suppose you need to locate a middle point between two given points which are not the end points of a line segment, what will you do? There is no direct operation of using the object snap directives to locate the point in **TwinCAD** (why there is no such built-in function provided by the system is another story), so you have to do it indirectly.

If you are not aware of the power of TCL expressions, the common solution to you is to draw an auxiliary line segment connecting the two given points and then use the object snap directive MID to locate the middle point of the line segment to the command prompt that requires the data point. Additional operation may be required to erase the auxiliary line segment at later time.

You no longer need to go through all the trouble finding the solution. You may take full advantage of TCL expressions and solve it by entering the following expression to the command prompt each time when you need to locate the middle point:

```
prompting message...: (getp()+getp())/2
```

The system will then ask you to designate the two points, as each required by the **getp()** function, and then the middle point between them is calculated. If you feel you may likely pick up at the wrong points and you want to have a chance to correct them before returning the data point to the prompt, you would appreciate this alternative solution given below:

```
prompting message...: 'p1=getp() <pnt1>  
prompting message...: 'p2=getp() <pnt2>  
prompting message...: (p1+p2)/2
```

The first two expressions are given in transparent mode (prefixed with an apostrophe) and the points obtained by the two **getp()** functions are stored in the TCL registers P1 and P2 respectively. Before you entering the last expression, which calculates the middle point between P1 and P2 and returns the result to the prompt, you can assure that P1 and P2 are containing the correct data points. Since, If not, you may re-enter the transparent expressions again.

If you need this operation so often, you may put these expressions as menu scripts in the menu file to associate with a pulldown menu or a screen menu item, so that you may invoke it each time you need it. You may like to add prompting message to the **getp()** functions as its first argument, if they are invoked from the menu scripts. See the example at the next section.

Extending TwinCAD with TCL

The usage of TCL expressions in previous example may be further implemented into a TCL program that can be loaded by **TwinCAD** as an extension part of the system. An example implementation is listed below:

```
FUNCTION POINT mid()  
{  
    p1=getp("\nFirst Point:");  
    p2=getp("\nSecond Point:");  
    return (p1+p2)/2;  
}
```

You may use the command **CLOAD** to load in this small program. Once this program is loaded (the system will report that the function **mid()** is defined at the loading), it will become a part of the system (just like the **getp()** function). And then, you may issue the function **mid()** each time you need to locate the middle point between two given points to the command prompt, as the example below:

```
prompting message...: mid() <enter>  
First Point: <pick a point>  
Second Point: <pick a point>  
operation continue...
```

The **mid()** function is then an extended function to **TwinCAD**. You may setup the system to load in this TCL program automatically at the system start-up, so that the **mid()** function can be taken as if it is a built-in object snap function from the system.

You can also extend the command set of **TwinCAD** by loading the TCL programs that define commands. For example, you may like to have a command called STAR to draw stars of specific size at designated places. The following is an example implementation:

```
COMMAND Star()  
{
```

```
int    i;
double size;
POINT  center;
POINT  vertex[5];
size = 5.;
center = 0,0;
for(;;)
    center = getp("\nIndicate where to put the star: ",
                 center);
    size = getv("\nSize of the star(%8.3f): ",size);
    for(i=0;i;i++) // Find vertex points
        vertex[i] = center + p(size,(90+i*72)A);
    command("line",
            "\@vertex[0]",
            "\@vertex[2]",
            "\@vertex[4]",
            "\@vertex[1]",
            "\@vertex[3]",
            "close");
    }
}
```

Again, use the **CLOAD** command to load in this program and you will see that a new command named STAR is defined. Now, to draw a star is a simple thing as the command STAR has become a part of the system. You may enter the command STAR to draw stars as the example sequence given below:

```
@CMD: STAR (return)
Indicate where to put the star: (pick a point)
Size of the star( 5.000): 10 (return)
Indicate where to put the star: (pick a point)
Size of the star( 10.000): (return)
Indicate where to put the star: ^C
@CMD:
```

You may include this TCL program into **TCAD.TCL**, so that the **TwinCAD** will load it in automatically at start up time. More complicated and useful TCL program examples are given in the PART 4 of this reference manual.

Writing TCL Programs Using Text Editor

TwinCAD does not provide a general text editor with the package for writing TCL programs. You may write your TCL programs using the text editor or word processor you like supplied by other vendors.

You don't need to exit the **TwinCAD** to access your text editor for TCL programming. Use the **DOS** command to invoke your editor. In fact, a general programming activity may be looked like this:

```
@CMD: DOS
C:\TCAD>edit tclfile
C:\TCAD>
@CMD: RUN tclfile
```

@CMD: UNDO

...

where *edit* is the program name of your text editor, and *tclfile* is the TCL program file. The **RUN** command is used to load in a TCL program and execute it immediately from the first function body (main program body), and then release the program after it finishes. You may undo the result of the TCL program execution by the **UNDO** command, if necessary.

Note that if you are implementing an extended function, you may like to replace the **RUN** command with **CLOAD** command and then using TCL expressions to test the function. You must use the **UNDO** command to unload the TCL program if you must modify it and load it in again.

Protecting Your TCL Programs

Your TCL programs are saved and retrieved as ASCII text files. Any one who has a copy of your TCL program files can use and modify them as much as you can. So, a TCL program file is unprotected by nature.

However,4(ile12. t)-13.4(ecd(l)-9(e)-0.((pr)-6.2ile12.i3(m)-24.e(o)-0.ons)-7.92(w)9. th)-12.2(en)-12(y)12((Yo)-12.2

>listfile optional, specifying to redirect the listing output to the listfile.

The current release of TCL.EXE (Version 1.10) will generate a simple listing of the source program, containing effective line number used for identification of the offending line where an error occurs. Fully documentation about this TCL.EXE utility will not be given at present time, since its implementation is not yet stable, and is subject to change in near future. One thing for sure, is that the final goal of this TCL.EXE utility is a TCL compiler that will produce threaded codes executable to the TCL Interpreter's kernel.

The TCZ file

The TCZ file, generated by TCL utility, is a '*compiled*' version of the TCL program file. It can be loaded, executed as the TCL program file, but it can not be edited by any means. The loading of a TCL program file may require to load in other files as specified by the **#include** preprocessor directive, while a TCZ program needs no other source include files during the loading.

The current TCZ file translated by TCL utility is in the program file format of TCL Source Application Level 1. The loading speed of such TCZ files will be faster than the original TCL source program files, but the execution speed is the same.

You may distribute your application programs in TCZ file format, which can not be read nor be edited by other people, so that your works are protected.

Binding Application With TwinCAD

The TCL library functions provides three functions for you to bind your TCL application program with specific copy of **TwinCAD**. They are **series_no()**, **card_mid()** and **card_zid()**, for you to read the system's series number, hardware protector's master ID. number and zone ID. number respectively. See the part of Runtime Library function support for more description to these functions.

To bind your TCL application program with a specific copy of **TwinCAD**, follow the procedure below:

1. Read the series number or hardware protector's master ID. number from the specific copy of **TwinCAD** by issuing **series_no()** or **card_mid()** to the command prompt. (You may call your customer to do that for you before shipping the TCZ programs.)
2. Modify your TCL source programs with those informations read from item 1 (see example below).
3. Use TCL.EXE to translate your TCL source programs into TCZ file format.
4. The TCZ file now is bound with the specific copy of **TwinCAD**. You may distribute it to the user of that copy of **TwinCAD**.

An example of TCL program that checks the series number and master ID. number is given below:

```
#define SERIES "XXXXXXXXXXXX"  \\ change this by user's no.
#define MID    0L
COMMAND TEST()
{
    if ( (series_no()!=SERIES) || (card_mid()!=MID) )
        printf("\nUnauthorized Access!");
        exit(1):
}
...
```

}

TCL Lexical Elements

Operators and separators

The operators in TCL may be a single character or multiple characters, such are:

```
! % ^ & * - + = ~ | . > < / ?  
++ << >> <= >= == != && || += -= *= /=  
%= <<= >>= &= |= ^=
```

The parsing of the operator requires no intervening spaces in a multiple characters operator.

The separators are:

```
( ) [ ] { } , ; : white-spaces
```

The **()** are used to group expressions, **[]**, to specify array element, **{ }**, to group statements, **,** (comma) to separate expressions, **;** to separate statements, **:** to ends a label (also used in conditional expression), and white-spaces are used to separate tokens.

Identifiers

An identifier is a name in TCL. It is a sequence of letters, digits, underscores '_' and dollar sign '\$' (Standard C does not accept dollar sign '\$' as a legal character for identifier). It must start with a letter, underscore or dollar sign character, and must not have the same spelling as the reserved words.

Note that the cases are sensitive for identifiers. The length of an identifier can be as long as 128 characters. But, if the identifier is a name of a function, then only the first **9** characters are used for the recognition.

Predefined Identifiers

Predefined identifiers are used to address TCL internal registers or system informations. Most of these identifiers are not part of the reserved word and may be overridden by local identifier declarations.

All predefined identifiers are described below:

TCL Registers	Any identifier starts with a character in the set { I, V, P, L, A, C, E, S } or in the set { i,v,p,l,a,c,e,s }, and followed by numerical digits, are taken as a direct register identifier. These are used to address the TCL internal registers, such as V1, P1, A9, C10, E0, and S2.
Type Function	Any identifier starts with a character in the set { I, V, P, L, A, C, E, S } or in the set { i,v,p,l,a,c,e,s }, and followed by a right parenthesis ')' are taken as a type cast of a predefined data type to evaluate the enclosed comma expressions to return the specific data type. It can be seen as a predefined identifier of function (intrinsic type functions), such as P(10,10), C(...) and V(P1,P2).
TCL Register Indexing	Any identifier starts with a character in the set { I, V, P, L, A, C, E, S } or in the set { i,v,p,l,a,c,e,s }, and followed by a right bracket ']' are taken as an indirect register identifier. The enclosed expression is evaluated to address the TCL internal registers by indexing, such as P[10] which is equivalent to P10.
'_arg_no'	The identifier '_arg_no' is predefined as a local variable to store the number of parameters actually passed to an invoked function. This identifier can not be overridden by local variable declaration.

System Variables

Any identifiers prefixed with a '@' is taken as a system-wide global variable. These are used to store and retrieve information to and from the system. There are many such predefined identifiers with different data types that control the operation of **TwinCAD**. Some of these informations will be saved with the work drawing. User application programs may also create such identifier via special function calls (*uservar()*). See the chapter of System Variables. Examples are **@curcolor** and **@DIMATYPE**. Note that both the upper and lower cases of characters after the '@' sign are accepted and treated the same.

Embedded Identifiers

Embedded identifiers are identifiers referenced in a string constant. Since a string constant is simply copied to the target system during compilation, an expression in a string constant can not be successfully evaluated by the *eval()* or by the command line interpreter if there are local identifiers being referenced. To force the TCL to resolve the reference to the local identifiers in a string constant during compilation time, the identifiers must be given in the form:

```
"...\@identifier..."
```

This is called the Embedded identifier.

For example, to create a line segment from the *StartPoint* to *EndPoint* using the **command()**, the expression:

```
command("LINE \@StartPoint \@EndPoint ");
```

has embedded the two local identifiers *StartPoint* and *EndPoint* in the string passed to the command line. Even though the identifiers are local variables and are valid only when the specific TCL program is active, the command line interpreter may still access these variables correctly from the expression.

Reserved words

The following identifiers are reserved in the TCL language, and must not be used as program or variable identifiers.

auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while, typeof, POINT, LINE, ARC, CIRCLE, COMMAND, FUNCTION, ENTITY, LAYER, LTYPE, SLIST, ENTDATA, STYLE, FILE, ELLIPSE.

Some of these reserved words may not be fully supported in current version of TCL, such as **enum, struct** and **union**, or may be meaningless in current implementation, such as **volatile, static, auto, extern** and **const**.

Constants

There are four kind of data constants; namely, they are integer constant, floating point constants, string-constants, and character constants.

Integer constants

The TCL uses the two's-complement representation for integers. The storage of an integer can be 16-bits or 32 bits, depending on its type declaration or on its value. The following table lists the supported form of an integer constant:

Radix	Prefix	Body	Postfix
* Binary	None	binary string	'b'
* Octal	None	octal string	'o'
Octal	'0'	octal string	None
Decimal	None	decimal string	None
* Hexdecimal	Digit	hexdecimal string	'h'
Hexdecimal	'0x'	hexdecimal string	None
Hexdecimal	'0X'	hexdecimal string	None

An optional long marker 'L' or 'l' can be used to specify the 32 bits storage to store the constant. Note that, TCL always use 32 bits long integer to store the evaluated result internally, if it should fall off the 16 bits range.

Items with a '*' mark are not standard for C language, but was supported by the original expression evaluator of **TwinCAD**, which is also the kernel expression evaluator of all **TCAM** series products.

Floating point constants

Real number are always expressed in decimal radix. Scientific notation of real value with exponent part is accepted. Generally, the number is expressed in the following format:

{-} {integer_part} '.' {fractional_part} {e|E|d|D}{+,-}nnn}

The *integer_part* and *fractional_part* are a series of decimal digits. An example is -123.456E-7, which is equivalent to -0.0000123456 without the exponent part. All real numbers in TCL are in IEEE 64-bits floating point format.

Character Constant

All character constants have the type int. Their values are the integer encoding of the ASCII characters. A character constant may be in the form:

- 'c' Direct ASCII value of the character code c.
- '\c' Character Escape Code if the character c is a valid part of the character escape codes; otherwise, it is equivalent to 'c'.
- '\nnn' Specifying the code in octal constant, where nnn represents the octal value. For example, '\0' means null character, and '\015' is the carriage return code.

Character Escape Codes

Character escape codes are used to represent special control characters. Character escape codes are introduced by a '\ backslash, as in the list below:

- \n a new line code
- \t a horizontal tab code
- \b a back space code
- \r a carriage return code
- \f a form feed (clear screen) code
- \\ a backslash code

<code>\"</code>	a double quote
<code>\'</code>	a single quote
<code>\nnn</code>	a code in octal constant, where nnn is the octal digit string. Such examples are <code>\015'</code> and <code>\012'</code> for carriage return and newline codes.

All else characters after the `\` will be taken as what it is.

String constants

A string constant is in the form:

"char string"

The type of a string constant is an "array of char". The storage allocated to store a string constant is the length of the string plus one. All strings are terminated by a null character automatically. Any characters can be included in the string constant, except the null code, which is used as a string terminator internally, and the new-line character, which terminate the string physically during the preprocessing. Use the Escape Character Codes to represent these special control codes in a string constant.

TCL Registers

Not only being a language interpreter, but also served as a virtual machine, the TCL has provided predefined objects called TCL Registers. TCL Registers are system-wide storages, global to all TCL programs, directly accessible from the command line, used for temporary storage, quick accessing and data exchange. The following lists out the equivalent TCL declaration of these TCL registers:

```
int      I[50];      // From I0 - I49 or i0 - i49
double   V[50];      // From V0 - V49 or v0 - v49
POINT    P[20];      // From P0 - P19 or p0 - p19
LINE     L[10];      // From L0 - L9 or l0 - l9
ARC       A[10];      // From A0 - A9 or a0 - a9
CIRCLE   C[10];      // From C0 - C9 or c0 - c9
ENTITY   E[20];      // From E0 - E19 or e0 to e19
char     S[10][128]; // From S0 - S9 or s0 - s9
```

Note that

- Both upper case and lower case characters are accepted to address these registers.
- Though they are declared as arrays, the single character of {I, V, P, L, A, C, E, S} will not be taken as an array of registers.
- Index addressing is possible by the expression like `Vn[idx]`. The addressed register will be the (n+idx)'th of the V register in the example.
- Array addressing is implicitly done by `&Vn`, when it is passed to the runtime functions.
- String register `S0 - S9`, can not be addressed to individual bytes, since the expression `S0[4]` will be taken as index addressing of S register.
- TCL registers may be hidden (overloaded) by local TCL variable declarations.

The use of TCL registers is very convenient to the beginning user, who needs not to know how to declare variables before writing expressions using them. Also, at the command line, the operator may directly access them for temporary storage using TCL expressions.

Special Note on Backslash Code in String Constant

The backslash code used in a string constant serves as an escape code introducer. That means, the code itself will not be treated as a data code, unless it is repeated. This should be no problem with TCL programmer writing the text string in English or in 8 bits IBM-Extended ASCII characters. However, if the programmer is writing in Chinese, Japanese, Korean and the like, he is likely to run into trouble due to the code conflict.

A Chinese character consists of two bytes. The first byte is always higher than $0xA0$, so as to be distinguished from the ordinary 7-bits ASCII. The code range of the second byte covers the whole 8-bits ASCII except those control codes. The backslash code is therefore a valid code for the second byte of a Chinese character. As a consequence, if a Chinese character containing a backslash code as its second byte, the byte must be repeated in the text string; otherwise, the lexical parser will remove the code and convert the code after it according to the rules stated earlier, and thus the text string is distorted.

Usually, the programmer is not very easy to be aware of such situation. Checking a text string displayed in Chinese from a text editor to see if it contains a misleading backslash code is a tedious work.

The TCL interpreter has solved this problem by adding a preprocessor directive that enable/disable the recognition of two-byte codes in a string constant. However, as the Standard C/C++ does not support this kind of feature, the program written in such a way may still cause problems in the future when the programmer needs to port it to a real C/C++ language compiler.

Another solution is to supply a utility that will scan the programs and make the modification (adding backslash codes) automatically. Such a utility is useful not only to TCL programs but also to C/C++ programs. However, as most text books would say, this will be left as an excise!

See TCL Preprocessor Directive section for further details.

The TCL Preprocessor

The TCL preprocessor is a simple macro processor that processes the TCL source text before the compilation. It also tokenizes the input source, remove the comments, and checks the balance of parentheses, blankets, and braces.

The preprocessor can be controlled by special preprocessor command lines, which are lines beginning with a '#' in the source file, containing preprocessor directives.

The following lists the supported preprocessor directives.

#define	Define a preprocessor macro. Parameters and nested macro expansion are supported.
#include	Include in a file during compilation. Nested include files up to 10 levels are supported.
#ifdef	Test if a macro symbol is defined or not for conditional program compilation.
#ifndef	Test if a macro symbol is not defined or not for conditional program compilation.
#else	Same usage as above.
#endif	End of the IF/THEN/ELSE construct.
#ifyes	Ask for confirmation for conditional compilation.
#outmsg	Emit message during compilation or preprocessing.
#codechk	Enable/Disable checking of extension codes in string constant.

The following sections will describe them in details.

#define

The **#define** directive is used to define a symbol or a macro substitution. It has two forms, depending on whether or not a left parenthesis '(' is immediately following the name to be defined. The simpler form without a left parenthesis is:

```
#define   name   sequence-of-tokens
```

A macro defined in this form takes no arguments. The name is defined as a symbol to represent the sequence-of-tokens. Whenever the name is encountered in the source program text, the name is replaced by the definition body, i.e, the associated sequence-of-tokens.

The name must be a valid TCL identifier. If the first character of the definition body is an alphabet, digit, or left parenthesis, then at least a space character is required to separate the name with the definition body, which is the rest of the line. Note that the syntax does not require an equal sign or any other special delimiter token after the name being defined.

Examples of simple macro definitions are:

```
#define T_LINE 2
#define T_ARC 3
```

which makes the expression

```
if ( etype == T_LINE ) ...
```

much more readable than

```
if ( etype == 2 ) ...
```

The more complex form of a macro definition that takes arguments is given below:

```
#define name(name1,name2,...,namen) Sequence-of-tokens
```

The left parenthesis must immediately follow the name of the macro name without any intervening white-space. The names of the formal parameters within the pairs of parentheses must be identifiers, unique to each others. The definition body may or may not contain the reference to the formal parameters.

When such a name is referenced in the program text, the name must be in the same form: followed by a left parenthesis, actual argument token sequences corresponding to each of the formal parameters, and then a right parenthesis. The actual argument token sequences are separated by commas. Each of the arguments is used for the individual substitution of names in the macro body definition.

For example, when the macro definition

```
#define sum(a,b) ((a)+(b))
```

is invoked by the expression

```
sum(x+2,y-3)
```

the macro substitution will produce the expression

```
((x+2)+(y-3))
```

as a result, and also the expression

```
sum(sum(x+2,y-3),x*y)
```

will become

```
((((x+2)+(y-3)))+(x*y))
```

as the result of the nested text substitution.

Notice the use of parenthesis in the definition body in the above examples. It is not necessary to parenthesize the name of formal argument within the definition body. However, if the text substitution may produce result of ambiguity, then parentheses can be used to clarify the problem. For example, under the same macro definition without parentheses:

```
#define sum(a,b) a+b
```

then the expression

```
sum(x+2,y-3)
```

will become

```
x+2+y-3
```

but the expression

```
z*sum(x+2,y-3)
```

will become

```
z*x+2+y-3
```

which is quite different from the expected meaning when the programmer wrote the $z*sum(...)$ expression. In such a case, the parentheses must be used in the definition to solve the problem.

Nested Macro Expansion

Once a macro is invoked and causes the text line be expanded, the expanded line is then rescanned for further macro expansion until no more macro expansion is encountered. However, the expansion will be stopped if the length of the resulting text line will exceed the maximum limits of 255 bytes (after tokenization).

Note that the text line after macro expansion will not be checked for the preprocessor directive, even though it might contain one. The preprocessor directive is checked before the macro expansion.

Macro Substitution on Embedded Identifiers

The *Embedded Identifiers* in a string constant are recognized and checked for the macro substitution before their reference are resolved. For example, the program fragment below:

```
POINT  pstart, pend;
...
command("line \@pstart \@pend ");
```

can be modified to utilize the TCL registers with the preprocessor **#define** directives:

```
#define pstart p1
#define pent   p2
// POINT  pstart, pend; // this line is commented out
...
command("line \@pstart \@pend ");
```

The programmer needs not to change the embedded local identifiers (variables pstart and pend) to TCL registers, as the TCL preprocessor will handle the macro substitution for them.

Note that practice of complexed or nested macro substitution of the embedded local identifiers is not recommended, as the preprocessor must recognize the embedded local identifier by its prefixing '@' character string and the macro substitution will deprive the identifier of its prefixing '@' string.

Predefined Macro

TCL has predefined a macro named "TCL" for the identification of the TCL language processor.

#include

The **#include** directive is used to include a separate file into the section for compilation, as if the content of the file had appeared in place of the **#include** directive. There are two forms:

```
#include <file>
```

or

```
#include "file"
```

No file extension or data path are assumed for the specification of the include file. The included file may also contain preprocessor directives to include another program files. The include files may be nested down to 10 levels. Note that recursive file inclusion is not allowed and will be detected as an error.

If the include file can not be found at the directory specified, the system will search the current directory for it. The directories specified in the environment variable **TCADPATH** will also be searched through if the file can not be found in the current directory.

#ifdef and #ifndef

The directives **#ifdef** and **#ifndef** can be used to test whether a name is defined as a preprocessor macro. They are used in the form:

#ifdef *name*

and

#ifndef *name*

If the name is defined and tested by the **#ifdef**, or the name is not defined and tested by **#ifndef**, a true condition is resulted. The program texts after the preprocessor line will be processed until the end of file, or another **#else** or **#ifxxx** directives to change the state of processing.

If a false condition is resulted after the test, the program texts after the preprocessor line will not be processed until the end of file, an **#endif**, or an **#else** directive is encountered.

These two directives are used for conditional compilation. Note that during the conditional compilation, if the state of compilation is off, then only the preprocessor directives of conditional compilations are processed, all else text lines are skipped.

#else and #endif

These two are also directives of conditional compilations. The directive **#else** is used to change the state of conditional compilation, while the **#endif** is to terminate a section of conditional compilation.

#ifyes

The directive **#ifyes** is also used for conditional compilation. It is used in the form:

#ifyes *string*

or

#ifyes "*string*"

When the TCL preprocessor encounter this directive, if the state of compilation is ON, it will prompt the string to the standard output and wait for the operator to answer 'Yes' or 'No' by entering the character 'Y' or 'N' from the keyboard. The default is always 'Yes' when the operator presses the return key directly.

If the operator returns 'Yes', then a new section of conditional compilation is setup with a state of ON; otherwise, the compilation state will be OFF for the section.

#outmsg

The directive **#outmsg** is used to emit message to the standard output during the program text file compilation. It is used in the form:

```
#outmsg    string
```

or

```
#outmsg    "string"
```

where the **string** is the message to output.

#codechk

The directive **#codechk** is used to enable or disable the lexical parser to check the two-bytes extension codes within the string constant. It is used in the form:

```
#codechk    start
```

where **start** is an integer constants to specify the starting range of the leading code of the two-byte codes. If the it is less than 0x80, then the checking will be disabled. If the starting range is missing, the checking will be disabled too. The ending code range is always 0xff.

When the code checking is enabled, the lexical parser will parse the string constant in two-byte codes. That means, starting from the first byte of the string, if the value of a byte is in the range specified, then the next byte will be passed directly without backslash code checking.

An example of this directive to turn on code checking for Chinese Big-5 code is given below:

```
#codechk 0xA1
```

Once it is effective, the program may turn it off anytime by:

```
#codechk 0
```

The initial code checking default is off, so as to be compatible with ANSI C/C++.

Examples

An example of TCL program fragment containing macro usage is given below:

```
#outmsg "\nGenerate plotting of surface(x,y) = exp(-x*x-y*y)"  
#ifyes "\nNo real generation, plot only (yes/no) (Y) ?"  
    #define newent(x)  plotent(x)  
#endif  
#define exp(x,y)  exp(-(x)*(x)-(y)*(y))  
...  
dx = 0.2;  dy = 0.2;  
for(x=-2;x<=2;x+=dx)  
    p1 = P(x,-2)*100;  
    z1 = exp(x,-2)*100;
```

```
    for(y=-2+dy;y<=2;y+=dy)
        z2 = exp(x,y)*100; // Calculate exp() by macro
        p2 = P(x,y)*100;
        L1 = L(p1,p2,z1,z2);
        newent(L1); // Create or Plot, depends
        p1 = p2;
        z1 = z2;
    }
}
...
```

Declarations

To declare an identifier in TCL language is to associate the identifier with the TCL variables or functions. There are three kind of declarations supported in TCL; namely, they are:

- **Simple Variable Declarations**
- **Array Variable Declarations**
- **Function Declarations**

This chapter will describe how a variable or a function can be declared for use. Included in this chapter are also those predefined **typedef** data types.

Variable Declarations

All variables must be declared before use, although certain storage of objects (like TCL registers and system variables) can be accessed without explicit declaration. Declarations are required to associate identifiers with the variables to be used in the program, to specify the data type they have, and optionally, to assign their initial values.

Some typical variable declarations are given below as examples:

```
int count=0, lower, upper, step;
char c, buf[256];
char message[]="\nPlease pick an object: ";
double matrix1[3][3], matrix2[3][3];
POINT origin={0.,0.}, center;
```

The general form of declarations are given below:

{Class} {Type} Identifier{'=Initial-value'}{,Identifier...};

for simple variable declaration, and

{Class} {Type} Identifier['exp']{['exp']...}{'=Initial-value'};

for array variable declaration. Note that although all the storage class specifiers (static, extern, auto, register, const) of C language are reserved in TCL, they are merely accepted and mean nothing in current version. A data variable declared in TCL will assume the storage class of auto if it is declared inside a function body, and assume the storage class of static if it is outside a function body.

Scope of Declaration

The scope of a declaration is the region of the program text over which the declaration is active. The scope of an identifier declared as a variable is governed by the following rules:

- An identifier declared outside of a function definition has a scope that extends from its declaration point to the end of the source program file.
- An identifier in a formal parameter declaration has a scope that extends from its declaration point to the end of the function body.
- An identifier declared in a function body has a scope that extends from its declaration point to the end of the function body.

A declaration of an identifier is visible through out its scope, but it may be hidden by other declaration of the same identifier with an overlapped scope. Such an example is given below:

```

int A;
fnc1()
{
    double A, B;
    ...
}
fnc2()
{
    ...
}
    
```

The integer variable *A* has a scope cover the both functions *fnc1()* and *fnc2()* in the above example, while the double variable *A* and *B*, has a scope within the body of *fnc1()*. So, the integer variable *A* is visible to *fnc2()*, while the double variable *A* and *B* are not visible to it. As the double variable *A* is declared locally in the function body of *fnc1()*, the integer variable *A* is hidden from the *fnc1()*.

Data Type and Type Specifier

The TCL supports the following elementary data types:

Char	Signed, 1 bytes
int, short	Signed, 2 bytes (16 bits)
long	Signed, 4 bytes (32 bits)
float	8 bytes IEEE-64 bits floating point value (treated as double)
double	8 bytes IEEE-64 bits floating point value

All integer values are signed values in current version of implementation, though the keyword unsigned is reserved.

To facilitate the CAD/CAM application programming, TCL also supports several predefined data type of **typedefs**. These predefined data types are structured objects and are directly supported by the runtime library as well as by the TCL expression evaluator.

The following list the predefined **typedef** data types:

POINT	Structure of data points with X/Y component.
LINE	Structure of 3-D geometry lines.
ARC	Structure of 2-D arc segments.
CIRCLE	Structure of 2-D circles.
ELLIPSE	Structure of 2-D ellipses or elliptic arcs.

ENTITY	Structure of a entity handle, containing information about existing drawing entity.
ENTDATA	Union of different kind of entity data structure, containing geometry information of the specific entity.
SLIST	Pointer to a dynamically allocated array of entity handle (ID. only).
STYLE	Structure of a loaded text style table entry.
LAYER	Structure of a created layer table entry.
LTYPE	Structure of a created linetype table entry.
FILE	File handle (equivalent to int).

The details of the **typedefs** will be discussed later in this chapter.

Initializer

The declaration of a variable may contain an initializer that specifies the initial value of the variable at the beginning of its lifetime. The initializer after the equal sign '=' in the declaration are a series of expressions that can be evaluated to the type of the variable it is to initialize. Each expression, separated by a comma, corresponds to each element of the variable in declaration. Type conversions are done automatically. The expressions may be arbitrary, as long as the TCL can evaluate it at the time it is encountered.

For example, the following program fragments are legal:

```
double  x=10, y=20.;
POINT  pz={x,y}, px=30,40;
double  z={x+5};
CIRCLE cc=C(p,x+30);
```

Note that the initializer may contain braces to enclose the expression to make it more readable. As a matter of fact, in the above example, the braces are also necessary to initialize the POINT data; otherwise, the lexical parser will take the x to initialize the p, and declare y as a POINT data variable.

The general rules about the initializer are given below:

- Complex expression can be used as the initializer. This includes function calls (e.g., the variable *cc* in the above example).
- The expression in the initializer may contain reference to variable. However, forward reference is not allowed.
- If comma expressions are used to initialize a single variable, braces must be used to group them together (e.g. variable *pz* in the above example), unless these expressions contain no reference to variables and it is the last variable in the declaration (e.g. variable *px* in the above example).
- If the initializer expression should contain reference to variable, it should be enclosed in the braces, even if it is a single expression.

The use of complex expression in data initializer is convenient, but is not compatible with Standard C/C++ syntax, and is therefore not recommended.

Declaration of formal parameters may not have initializers. Special data type like SLIST, ENTITY, ENTDATA, STYLE, LTYPE and LAYER, may not have initializer.

Array Variable Declarations

TCL accepts multidimensional array up to 64 dimensions. However, the total number of element count for an array variable must not exceed 65535. The number of elements in an array dimension is determined by the expression in the enclosed bracket **[exp]**. The expression can be arbitrary expression that can be successfully evaluated to a positive integer value.

The expression to determine the size of an array dimension can be omitted as long as it is not needed to allocate storage when:

- The object being declared is a formal parameter of a function.
- The declaration contains an initializer from which the size of the array can be determined.

Examples are given below:

```
char msg[]="A character array!";  
int order[]={4,1,2,5,6,3};
```

An exception to these cases is the declaration of an N-dimension array, which must include the explicit size of the last N-1 dimensions so that the accessing algorithm can be determined. For example:

```
char wday[][4]={"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",  
"Sun"};
```

When an array declaration is encountered, TCL will allocate the required storage from the TCL data segment, and initialize it according to the initializer. Multidimensional arrays are stored such that the last subscript varies most rapidly. That is, the element of the array

```
int t[2][3];
```

are stored in the order as

```
t[0][0], t[0][1], t[0][2], t[1][0], t[1][1], t[1][2]
```

The expression in the inner of **[...]** to address the element can be a comma expression, which is used to subscript a multidimensional array. Therefore, the expression

```
array[i,j,k]
```

is equivalent to

```
array[i][j][k]
```

in TCL! Note that this is different from the Standard C/C++ language. For portability consideration, it is not encouraged to write such expressions in array subscription.

Note also that TCL will perform bound checking about the subscript to an array element against the declared value. Should a subscript value out of the bound, a runtime error will happen.

Function Declarations

The general form of syntax to declare a function definition is given below:

```
{Class} {Type} Identifier '('{parameter-list}' compound-statments
```

Class of Function

There are three classes of function can be declared; namely, they are

- Static** Declare the function to be known locally only. This is the default. All functions declared without class specifier or with other C standard specifier are taken to be static in TCL, in current version.
- COMMAND** Declare the function to be an entry point from the TwinCAD command line. The name of the function will become the new extended command. Note that character cases are not checked by the command line handler and no argument will be passed by the command line handler. A recommended convention is to capital the first letter of the identifier.
- FUNCTION** Declare the function to be a global function to the system. Such a function, once loaded, can be referenced in any expression either from the command line or from another TCL program, as if it is an internal runtime library function. Note that, since all variables are local to a TCL program and are allocated during the runtime, the function body may only reference the variable locally defined in its body. This is because during the runtime, only the function body is executed when the function is invoked.

Examples are given below:

```
//    Report current layer information
COMMAND Clayer()
{
    LAYER    curlayer;
    curlayer=clayer();
    printf("\nCurrent layer: %s, Ltype: %s, Color: %d",
        curlayer.name, curlayer.ltype, curlayer.color);
}
FUNCTION double factor(n)
int    n;
{
    double    dval=1;
    while(n>1)
        dval *= n--;
    return dval;
}
```

Type of Function

All the data type (include predefined **typedefs**) can be used to declare the data type of a function to return. A special data type, **void**, is also supported for a function to return nothing.

If a function is declared with a specific data type, then the returned value will be checked against it. If it is of different type, type conversion will be done automatically. However, if the returned value can not be converted to the type specified, a runtime error happens.

If the type specifier in a function declaration is omitted, then the return value will not be checked anyway, and the function will assume the data type as its return value. This is useful if a function should accept different parameter and thus return different type of data; nevertheless, it is not encouraged to do so.

Parameter List in Function

There are two styles to declare the parameter list in a function. One is the older style in FORTRAN fashion, which places the declaration of parameter after the ')', like the example below:

```
double factor(n)
int n;
{
    ...
}
```

The other is in ANSI standard C style which places the declaration in the parameter list, like the example below:

```
double factor(int n)
{
    ...
}
```

TCL supports both styles. When a function is called and arguments are passed through the TCL data stack, TCL will automatically check the data type of these arguments against those declared parameters. If necessary, type conversion will be done automatically to match the arguments with the parameter list. If the matching fails, a runtime error happens.

The type declaration of a parameter in TCL function may be omitted. In such a case, the type checking and conversion process will be disabled for that parameter. It will assume the data type and size from the actual passed argument. The TCL function may use the cast function **typeof()** to obtain the data type of argument passed; however, it is not an encouraged style in writing a function.

There is one thing that is also apart from the C Standard. For an array argument, the declaration needs not to explicitly declare the size of the array dimensions, even a multiple dimension array, since the array is passed by an implicit pointer and the TCL is able to obtain the original specification of array dimensions. However, in standard C programming, only the left-most dimension of a multiple dimension array can be omitted. To maintain the compatibility, the TCL programmer may explicitly supply the size of the array dimensions in the declaration; yet, TCL will not check if the declaration matches with the argument or not. It is the original specification of the passed array argument used in the subsequent reference of the parameter in the function body.

Note also that in current version of TCL implementation, a TCL program may pass the argument in pointer expression (**&var**) to a function from the runtime library, but it may not use such expression to pass a pointer to a function written in TCL. This is because the TCL does not support the pointer type explicitly in current version. All arguments, except arrays, are passed by values. The only one exception to this is: the multiple dimension array of char type can be taken as an array of STRING type, and the string is passed by value (string itself). An example is given below:

```
char strtbl[12][4]={"Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
for(i=0;i<12;i++)
    if ( str == strtbl[i] )
        return i+1;
}
```

The **strtbl[][]** is a two dimension array of char, or a one dimension array of STRING. So the **strtbl[i]** reference to the i'th element of STRING, not characters. To reference to a single characters, use **strtbl[i][0]**. Note that the STRING is not a real predefined **typedef** data type in TCL, but a conceptual entity.

Predefined Typedef Data Types

The followings sub-sections describe these predefined data types. The name listed as a member of a structure is the part that a TCL program can directly address. Some structures containing the '...' mean that there are parts hidden from the TCL programs, and so that a valid object of that structure must be initialized by the built-in runtime function calls. There are hidden information which may not be accessible by a TCL program.

POINT Data Type

Declare the object to be a structure of 2-D data point with the X-component and Y component. An equivalent **typedef** in C language is given below:

```
typedef struct
    double x;      // X coordinate
    double y;      // Y coordinate
} POINT;
```

This is a most commonly used data type.

LINE Data Type

Declare the object to be a structure of 3-D line. An equivalent **typedef** in C language is given below:

```
typedef struct
    POINT ps;     // Starting point
    POINT pe;     // Ending point
    double zs;    // Z coordinate of starting point
    double ze;    // Z coordinate of ending point
} LINE;
```

ARC Data Type

Declare the object to be a structure of 2-D arc segment. An equivalent **typedef** in C language is given below:

```
typedef struct
    POINT ps;     // Starting point (read only)
    POINT pe;     // Ending point (read only)
    POINT pc;     // Center point
    double rad;   // Radius
    double span;  // Spanning angle in radians
    double angs; // Starting angle in radians
    double ange; // Ending angle in radians
} ARC;
```

Note that both the start point and end point of an ARC data segment is for read only. This is because there are no real fields in the memory storage for the data type. But whenever they are referenced for read access, TCL will automatically calculate the coordinate values and return them as if there are such fields.

ELLIPSE Data Type

Declare the object to be a structure of 2-D ellipse or elliptic arc. An equivalent **typedef** in C language is given below:

```
typedef struct
    POINT ps;     // Starting point
```

```

        POINT  pe;    // Ending point
        POINT  pc;    // Center point
        double rad;   // Radius
        double span;  // Spanning angle in radians
        double angle; // Rotation angle in radians
        double ecc;   // Eccentricity by axis ratio
        double angs;  // Starting angle in radians
        double ange;  // Ending angle in radians
    } ELLIPSE
    
```

The starting angle, ending angle and spanning angle of the ELLIPSE are taken from the parametric value to the following ellipse function in matrix form:

$$E(t) = rad * [\text{Cos}(t), \text{ecc} * \text{Sin}(t), 1] * [T]$$

where

$$[T] = \begin{vmatrix} \text{Cos}() & \text{Sin}() & 0 \\ -\text{Sin}() & \text{Cos}() & 0 \\ Xc & Yc & 1 \end{vmatrix}$$

and Xc, Yc are the center location of the ellipse, angle is the rotation angle of the ellipse with respect to the X-Axis about the origin, rad is the major radius, and ecc is the axis ratio. The length of the other axis can be calculated by rad*ecc. The value of ecc may be larger than 1.

CIRCLE Data Type

Declare the object to be a structure of 2-D circle. An equivalent **typedef** in C language is given below:

```

typedef struct
    POINT  pc;    // Center point
    double rad;   // Radius
} CIRCLE;
    
```

LAYER Data Type

Declare the object to be a structure of layer information. An equivalent **typedef** in C is given below:

```

typedef struct
    int    flag;    // Control flag
    int    color;   // Color of layer
    char   name[32]; // Name of layer
    char   ltype[24]; // Linetype name of layer
    ...
} LAYER;
    
```

You may obtain the layer information by the **layer()** and **clayer()** function calls.

LTYPE Data Type

Declare the object to be a structure of linetype information. An equivalent **typedef** in C is given below:

```

typedef struct
    char   name[24]; // name of linetype
    double unit;    // Base unit
    int    dash[12]; // Dash-dot specification
    ...
    
```

```
} LTYPE;
```

You may obtain the linetype information by the **getltype()** function calls.

STYLE Data Type

Declare the object to be a structure of loaded text style information. An equivalent **typedef** in C is given below:

```
typedef struct
    int     type;        // Type of style
    char    name[12];   // Name of style
    char    fname[64];  // Where it was loaded
    ...
} STYLE;
```

You may obtain the style information by the **style()** function calls.

FILE Data Type

Declare the object to be a file handle. An equivalent **typedef** in C is given below:

```
typedef int FILE;
```

ENTITY Data Type

Declare the object to be a structure of entity handle. An equivalent **typedef** in C language is given below:

```
typedef struct
    POINT  pick;        // Reference pick point
    ...          // Entity internal ID. etc.
} ENTITY
```

An entity handle is a structure containing information about a drawing entity from the database, which must be obtained by runtime library function calls, such as **gete()**, or directly assigned from other entity handle or entity selection list. The entity handle is referenced as a whole in accessing the drawing entity it points to.

A member of the entity handle '**pick**' is provided to store the pick point about the entity if it is picked by the operator via the **gete()**. The pick point of an entity handle is sometime very important to a CAD command in determining the way how the given entity will be processed. So, before supplying an entity handle to a command via the **command()** function call, make sure that the pick point is valid and meaningful to the operation.

SLIST Data Type

Declare the object to be a pointer to an array of entity handles. An equivalent **typedef** in C language is given below:

```
typedef ENTITY* SLIST;
```

This is a very special data type that is of pointer type implicitly. It provides a means to create an entity selection list that may contain variable number (unknown in advance) of entity handles. A TCL program must call **getesel()** or **ent_alloc()** functions to allocate the array of entity handles and to initialize them. Use **ent_count()** to determine the number of elements in the array, and use array subscript to address individual element.

An example program fragment showing the use of SLIST data is given below:

```
SLIST  slist;
```

```

int i, ecount;
slist = getesel("\nSelect object: ",0xffff);
ecount = ent_count(slist);
printf("\nTotal %d selected:",ecount);
e1 = slist[0];
if ( ent_ok(e1) )
    printf("\nOK");
for(i=0;i<ecount;i++)
    e1 = slist[i];
    process(e1);
}

```

Since SLIST is a pointer type, pointing to a dynamic array of entity handles, it must be initialized before use. However, in actual implementation, as the pick point of an entity handle is not used for a selection list, each element of a selection list is not really the same as an ENTITY type data. So, you can not assign an array of ENTITY type data to SLIST. The only way to initialize a SLIST data pointer is to use **ent_alloc()** or **getesel()** functions. See the descriptions of these two functions for more details.

These two functions will allocate memory from the local data stack as an array of modified ENTITY data, and assign it to an SLIST data pointer. To ease the TCL programming, there is no need to call another function to free this memory allocation when the selection list is not used any more. The TCL interpreter will automatically de-allocate the memory and make the SLIST data pointer invalid, when the function containing the call for its initialization exits.

This makes the handling of SLIST data much more easier in terms of memory management, but lays an important limitation on its usage: You can not return an SLIST data that is initialized from the returning function, since the exit of the function will de-allocate the data immediately. However, if you are necessary to write a TCL function that returns a selection list, you may require the caller to pass down the data memory as an argument. That is, the caller must use **ent_alloc()** function to allocate a data memory of a size that is supposed to be enough for the function application, and pass the allocated data memory (in SLIST data type, of course) to the function. Your TCL function application may use it to return the selection list. (You may utilize the **memmove()** to help the job.)

ENTDATA Data Type

Declare the object to be a union of structure of all kind of drawing entity data. An equivalent **typedef** in C language is given below:

```

typedef struct
int    flag;    // General entity flag
int    ltypeno; // Linetype ID of the object
int    layer;   // Layer ID of the object
int    color;   // Color of the object
union  edata
    struct epline  pline; // polyline
    struct epoint  point;
    struct eline   line;
    struct earc    arc;
    struct ecircle circle;
    struct eellipse elps; // ellipse
    struct eline3d line3d;
    struct etext   text;
    struct eblock  block;
    struct einsert insert;
    struct edim    dim;    // dimension

```

```
    struct eregion  region;  
    struct etag    tag;  
    struct esymbol symbol;  
    ...  
};    // Union specifier ignored deliberately  
...  
} ENTDATA
```

The union specifier is ignored in the above structure. To address the part of union, place the member of the union directly, such as `obj.line.ps.x` and `obj.layer`, letting `obj` be an object of the type **ENTDATA**. This makes the writing of the addressing to the final element shorter.

An **ENTDATA** type object must be initialized by **ent_read()** function call, which read the entity data from the drawing database. It is used in the retrieving-modifying-and-storing-back manner. To create a new entity, use **newent()** or **command()** function calls.

To guard from improper modification of the entity attributes, the data field **.flag** is for read only. The **ent_write()** function will not update the information from this data field. Useful bit flag values from this data field are given below:

- Bit 0:** Closed Polyline Flag. Valid for Polyline entity. If this bit flag value is ON, the polyline is closed; otherwise, it is opened.
- Bit 1:** Solid Fill State. Valid for closed Polyline, Circle, Ellipse and Region entity. If this bit flag value is ON, the entity is marked with a solid-fill state.
- Bit 2:** Tool-path Polyline Flag. Valid for polyline entity only. If this bit flag value is ON, the polyline is marked being used as a tool path for special CAM application purpose.
- Bit 3:** Open Tool-path Flag. Valid for tool-path polyline only. If this bit flag value is ON, the tool-path polyline is taken as an open path for special CAM application purpose.
- Bit 4:** Reserved.
- Bit 5:** Tag used. If this bit flag value is ON, the entity has been attached with tag attributes. You may use the **ent_tnext()** function to access the attached attribute tag entity.
- Bit 6:** ECS used. If this bit flag value is ON, the entity has its own Entity Coordinate System expressed by a normalized entity plane.
- Bit 7:** Region Flag. Valid only for BLOCK entity. If this bit flag value is ON, the BLOCK entity is actually a region definition.
- Bit 8:** If this bit flag value is ON, it means the entity was created or added by the system automatically. Such entity should be anonymous to the user.
- Bit 9:** Reserved.
- Bit 10:** Reserved.
- Bit 11:** If this bit flag value is ON, it means that the entity was erased.
- Bit 12:** If this bit flag value is ON, it means that wide line generation is used for that entity. The thickness of the entity will be used to specify the width of the line.
- Bit 13:** If this bit flag value is ON, it means that the entity is being frozen.
- Bit 14:** Blocked Element. If this bit flag value is ON, it means that the entity is a blocked element.
- Bit 15:** Polyline Segment. If this bit flag value is ON, it means that the entity is a polyline segment.

The structure of the members in the above union are described as below:

```

struct epline { // Type T_POLY
    double height; // Elevation of the polyline
    double thick; // Thickness
    ...
};
struct epoint { // Type T_POINT
    double x; // X coordinate
    double y; // Y coordinate
    double rad; // Radius, meaningless
    double height; // Elevation, meaningless
    double thick; // Thickness, meaningless
    ...
};
struct eline { // Type T_LINE
    POINT ps; // Starting point
    POINT pe; // Ending point
    double height; // Elevation
    double thick; // Thickness
    ...
};
struct earc { // Type T_ARC
    POINT ps; // Starting point
    POINT pe; // Ending point
    POINT pc; // Center point
    double rad; // Radius
    double span; // Spaning angle
    double angs; // Starting angle
    double ange; // Ending angle
    double height; // Elevation
    double thick; // Thickness
    ...
};

```

Note that the system manipulates the ARC entity in the drawing database by its start point and end point, which should be consistent with the definition of an ARC (equal distance to the center). As for the starting angle and the ending angle, however, they are ignored during operations. Conversions are done automatically from ENTITY data to ARC data type when an entity is in reference. Yet, the **ent_read()** and **ent_write()** functions may not check it automatically. It is therefore for the TCL programmer to maintain its consistency carefully.

```

struct ecircle { // Type T_CIRCLE
    POINT pc; // Center point
    double rad; // Radius
    double height; // Elevation
    double thick; // Thickness
    ...
};
struct eellipse { // Type T_ELLIPSE (T_ARC|0x10)
    POINT ps; // Starting point
    POINT pe; // Ending point
    POINT pc; // Center point
    double rad; // Radius
    double span; // Spaning angle
    double angle; // Rotation angle
    double ecc; // Eccentricity by axis ratio
    double height; // Elevation
    double thick; // Thickness

```

```

        double  angs;  // Starting angle
        double  ange;  // Ending angle
        ...
};
struct  eline3d { // Type T_LINE3D
    POINT  ps;  // Starting point
    POINT  pe;  // Ending point
    double  zs;  // Z-coordinate of starting point
    double  ze;  // Z-coordinate of ending point
    double  height; // Elevation, meaningless
    double  thick; // Thickness, meaningless
    ...
};

```

As internally the 2D-lines and 3D-lines are stored in different data structure in current version of **TwinCAD** (may be changed in future), A TCL program can not possibly read in a 2D-line and change it to 3D-line. However, a TCL program may erase the 2D-line and use the **newent()** function to create a 3D-line. When a line is created with same Z-coordinate values on both ends, it is automatically stored as a 2D-line.

```

struct  etext { // Type T_TEXT
    POINT  pbase; // Insert base point
    double  width; // Width factor used
    double  height; // Character height
    double  angle; // Starting direction angle
    double  obangle; // Oblique angle
    double  rad; // Radius for circular text
    double  tlen; // Text display length (RO)
    double  csp; // Character space
    int  styleno; // Standard style ID.
    int  estyleno; // Extended style ID.
    int  tadj; // Text Adjusting Flag
    char  str[72]; // Content
    ...
};
struct  eblock { // Type T_BLOCK
    POINT  pbase; // Insert base point (0,0)
    char  name[32]; // Name of block
    POINT  extmin; // Minimum extent coordinate
    POINT  extmax; // Maximum extent coordinate
    ...
};
struct  einvert { // Type T_INSERT
    POINT  pbase; // Insert base point
    POINT  pscale; // Scale vector
    POINT  pspace; // Spacing vector
    double  angle; // Insertion angle
    double  beta; // Array base angle
    double  gamma; // Second Array angle
    int  colno; // Column count
    int  rowno; // Row count
    char  name[32]; // Name of block
    ...
};

```

If both the **.colno** and the **.rowno** are one or zero, the INSERT represents a single block instance, and the array angle (**.beta**) and the element spacing (**.pspace**) will be meaningless. Otherwise, it represents a multiple array of block instances. However, if the most significant bit (bit 15) of the field **.rowno** is set to 1, it represents a rotational array of block instances. In this latter case, some of these data fields will have different meanings:

- .pbase** The center point of the rotational array.
- .pspace** The x component stores the angle increment between each element in the angle direction, and the y component, the radius increment in the radius direction.
- .angle** Initial insertion angle of the element at the starting angle position.
- .beta** The starting angle of the first element in the rotational array with respect to the center.
- .gamma** The starting radius of the first element in the rotational array with respect to the center.
- .rowno** The lower 12 bits (0xfff) stores the element count in the radius direction (the RINSERT command will always set 1 to this field in current release). The bit 14 (0x4000) is used to indicate whether to rotate the block instance around the center or not. If this bit flag is set, it means not to rotate.
- .colno** It stores the element count in the angle direction.

```

struct edim {     // Type T_DIM
    int  type;    // type of dimension and other inf
    int  flag;    // General flag of dimension
    int  atype;   // Arrow type used and others inf.
    int  iscale; // Scale factor in 8.8 format
    double unit; // Dimension value
    POINT ps;    // Dimension point
    POINT pe;    // Dimension point
    POINT pds;   // Extension point
    POINT pde;   // Extension point
    POINT pt;    // Dimension Text position
    char str[40]; // Dimension Text
    double angle; // Dimension Text Angle
    ...
};
    
```

For the field **.type**, only the lower four bits are used as the dimension type index:

0 Linear Dimension. Useful data fields are described below:

- .ps** The first dimension point.
- .pe** The second dimension point.
- .pds** The first dimension extension point. The first extension line is drawn from the first dimension point to this extension point.
- .pde** The second dimension extension point. The second extension line is drawn from the second dimension point to this extension point.

The position and direction of the dimension line(s) is determined by the dimension extension point. The angle between this dimension line and the first extension line determines the oblique angle.

1 Angular Dimension. Useful data fields are described below:

- .ps** The apex of the angle.
- .pe** The x and y components store the distances from the apex to each end point of the extension lines drawn, respectively.
- .pds** The first extension point, defining the start angle direction. It also defines the position of the dimension line (radius of the arc).

- .pde** The second extension point, defining the end angle direction counter-clockwise.
- 2 Diameter Dimension.** Useful data fields are described below:
- .ps** The first diameter point on the circle to dimension.
- .pe** The second diameter point on the circle to dimension. The middle point between the two diameter point is the center of the circle to dimension.
- .pds** If the dimension text is placed outside (as indicated by flag), this data field store the end point of the first extension line drawn from the second diameter point .pe. However, if the dimension text is placed inside and the Extension Arc Feature is effective, this data field and the next data field .pde will be used to store the two end points of the original arc.
- .pde** If dimension text is placed outside (as indicated by flag), this data field store the end point of the second extension line drawn from the .pds
- 3 Radius Dimension.** Useful data fields are described below:
- .ps** The center of the arc or circle for radius dimension.
- .pe** The diameter point on the arc/circle to dimension where to place the arrow pointer.
- .pds** If the dimension text is placed outside (as indicated by flag), this data field store the end point of the first extension line drawn from the diameter point .pe. However, if the dimension text is placed inside and the Extension Arc Feature is effective, this data field and the next data field .pde will be used to store the two end points of the original arc.
- .pde** If dimension text is placed outside (as indicated by flag), this data field store the end point of the second extension line drawn from the .pds
- 4 Ordinate Dimension.** Useful data fields are described below:
- .ps** The ordinate dimension point.
- .pe** The first intermediate point of the dimension lines.
- .pds** The second intermediate point of the dimension lines.
- .pde** The last end point of the dimension lines.
- If the bit 7 (Text Outside) flag is set, the dimension lines will be drawn from the .ps to .pe then to .pds and finally to pde. If the bit flag is not set, the dimension line will be drawn directly from .ps to .pde.
- 5 Leader Lines**
- The five data fields in the sequence, .pds, .pde, .pt, .ps and .pe are used to store the end points of the leader lines. The number of valid data points is stored in the lower byte of the .atype field. If the bit 7 (Text Outside) flag is set, it means that the leader entity has a valid dimension text and the field .pt will be used for the text position instead of a leader line end point.
- 6 Center Mark.** Useful data fields are described below:
- .ps** The center point.
- .pe** The x component stores the radius of the circle and the y component, the angle direction of the center mark.

- .pds** The x component stores the oblique angle of the center mark, and the y component, the scale factor for the center line on the oblique axis.
- .unit** This data field stores the size of the center mark.
- Else** Reserved, no image generation.

All else bit flag values from the **.type** are reserved and may have other special meaning. Currently, useful bit flag values that may be altered by the applications are:

- Bit 4 (0x10):** Special Text Option, currently valid only for linear dimension entity. If this bit flag value is ON, it means that the default text was and will be generated using the same rules for the diameter dimensions. The HDIM/VDIM/ALDIM commands will set this flag value if the user has picked up a circle to dimension.
- Bit 5 (0x20):** Oblique Feature Override. If this bit flag value is ON, then the oblique feature of a linear dimension will be turned off; that is, the dimension text and the dimension arrow will not be oblique anyhow.
- Bit 6 (0x40):** Fixed Text Angle Option. If this bit flag value is ON, then the **.angle** field will be used for the dimension text angle; that is, the dimension text angle will be fixed. If this bit flag value is OFF, then the dimension text angle will be determined dynamically for the default orientation.

For the field **.atype**, it is the upper byte used to specify the arrow types of the dimension, while its lower byte is used for other purpose. For example, for Leader dimension, the lower byte of **.atype** specifies the number of leader points. For other dimensions, the lower byte specifies the decimal precision for the default text generation (taken from DIMDPRE/DIMADPRE).

The upper byte of the field **.atype** is further divided into two nibbles (4-bit unit). Each nibble specifies an arrow type used for the dimension. The lower nibble specifies the first arrow type, and the higher nibble specifies the second arrow type. The use of these arrow types depends on the type of the dimensions:

- Both are used:** Linear dimensions, Angular Dimension and Leader Lines.
- Only one is used:** Diameter Dimension and Radius Dimension.
- None are used:** Ordinate Dimension and Center Mark.

Note that it is the arrow type 7 that turns off the arrow pointer, not type 0.

The meaning of these bit flag values in the **.flag** field are described below:

- Bit 0:** horizontal Attribute. On, if the dimension has a horizontal attribute. That is, it is an HDIM for a linear dimension, or a YDIM (dimension line in horizontal) for an Ordinate Dimension (if this flag is OFF, it will be an XDIM for an Ordinate Dimension).
- Bit 1:** vertical Attribute. On, if the dimension has a vertical attribute. That is, it is a VDIM for a Linear Dimension. Note that if both horizontal and Vertical Attributes are present, the Horizontal Attribute takes precedence. If both are not present, then it is an ALDIM for a Linear Dimension.
- Bit 2:** oblique Attribute for Linear Dimension. If this bit flag value is ON, it means that it is an oblique Linear Dimension.
- or Radius/Diameter Dimension, it marks for the Extension Arc Feature. If it is ON and the dimension text is inside (determined

- by flag value), an extension arc will be drawn to the point marked by .pe (Radius) or .ps (Diameter).
- Bit 3:** enter Text Option. If this bit flag value is ON, it means that the dimension text position .pt specifies the center position of the text. That is, the text will be center adjusted. If this bit flag value is OFF, the dimension text is always left adjusted.
- or Ordinate Dimension, this option specifies that the text will be half-height and left justified.
- Bit 4:** first Extension Line Suppression. If this bit flag value is ON, the first extension line will be suppressed. It is valid only for Linear and Angular Dimension.
- Bit 5:** Second Extension Line Suppression. If this bit flag value is ON, the second extension line will be suppressed. It is valid only for Linear and Angular Dimension.
- Bit 6:** Dimension Line Clipping Option. If this bit flag value is ON, it means that the dimension text is inside the extension line and the clipping of the dimension line is required. The clipping of the dimension line by the dimension text is not supported in the version before V2.10 (inclusive). This bit flag is valid only when bit 7 is OFF. Note that if this option is turn off, the dimension image generator will not check nor clip the dimension line against the text inside.
- Bit 7:** Text Outside Extension Line. If this bit flag value is ON, it means that the dimension text is assumed outside of the extension lines. The dimension lines will be drawn outside pointing inward on the extension lines. For Ordinate Dimension, if this bit flag value is ON, it means additional lead lines are used for the dimensioning and dimension lines will be drawn with 3 ld4(wil)-(ens)-8Tw [(wi-9.9(v [(wF)-6(.eg)-8.3on twF)-6(. Tf -8.5301 -1.7407 TD

For Angular Dimensions, this option is effective only when the OT option is OFF, and it will override the explicit given angle.

Bit 12: Dimension Line Reverse Option (OR) for Linear, Radius and Diameter Dimensions. See variable descriptions of DIMLOPT, DIMROPT and DIMDOPT for details on the Dimension Line Reverse Option.

Bit 13: Reserved. Don't change it.

Bit 14: Text Height Center Adjustment Option, valid only when Center Text Option (bit 3) is ON. It specifies to center the text by its middle point.

Bit 15: Fixed Text in Use. If this bit flag value is ON, it means that the text in the field .str is valid and will be used for the dimension text. If the bit flag value is OFF, then the dimension text will be fully associated with the dimension value and will be determined dynamically each time the dimension entity is generated.

For Leader Dimension, if the bit flag value is ON, then the dimension text is valid and effective and .pt will be used to specify the insertion point of the dimension text. In this case, at most four points can be used for the leader point specification.

```
struct eregion { // Type T_REGION
    POINT  pbase; // Insert base point
    POINT  pscale; // Scale vector
    double angle; // Rotation angle
    int    type; // Imposed type
    int    flag; // General flag
    char   name[12]; // Name of region
    char   patname[24]; // Name of pattern
    POINT  patbase; // Pattern Alignment Point
    double patscale; // Pattern Scale Factor
    double patangle; // Pattern Rotation Angle
    ...
};
```

The imposed status of the Region entity is stored in the **.type** field, which is currently to be one of the followings:

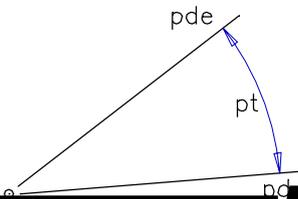
- 0** normal, Frame only.
- 1** attached with user specified lines. The specification of the hatch lines are followed (not present in the data structure).
- 2** attached with predefined pattern. The specification of the hatch pattern (by .patname, .patbase, .patscale and .patangle) is valid.

It is not allowed for TCL applications to modify the hatch specification directly, since the system needs to re-generate the associative hatching lines over the region when it is modified. The **ent_write()** will check to see this and will not update the following data fields to the drawing database: **.type**, **.patname**, **.patbase**, **.patscale** and **.patangle**. Note that those **.patxxx** fields are valid only when **.type** is 2.

To modify the associated hatch specification of a region entity, use **command("RHATCH...")** function call.

```
struct etag { // Type T_TAG
    POINT  pbase; // Insert point
    double width; // Width factor for text
    double height; // Height for text
    double angle; // Angle for text
    double obangle; // Oblique angle for text
```

```
double rad; // radius of circular text
double tlen; // Text display length (RO)
double csp; // Character space
int styleneo; // Standard style ID.
int estyleneo; // Extended style ID.
int tadj; // Text Adjusting Flag
char str[60]; // Tag content
char name[12]; // Tag attribute name
...
};
struct esymbol { // Type T_SYMBOL
POINT pbase; // Insert point
POINT pscale; // Scale vector
double angle; // Insertion angle
char name[18]; // Symbol name
double obangle; // Oblique angle for symbol
...
};
```



TCL Expressions

TCL, like C, is an expression language and has provided an unusually rich set of operators in its expression evaluator. The readers may find other appropriate books on C for a rigorous discussion on the syntax of expressions. In this chapter, we shall lay the emphasis on the discussion of operators and the difference from the standard C.

L-value

In later sections, the terminology **L-value** may be frequently referred to. So, it is better to explain what it is an L-value after all.

When a region of memory storage is allocated to store information, it is referred to as an object. An L-value is then an expression that refers to an object in such a way that the content of the object (information) may be altered as well as examined. In other word, an L-value can be placed at the left hand side of an assignment expression. Some of the TCL operators require L-value as their operands, such are the ++ and --. Of course, the left operand of an assignment operator must be an L-value.

Names of variables declared to have a storage class are L-values, except the name of an array (character array taken as a string can be an L-value in assignment). Register name is an L-value. But, the name of a function is not an L-values.

The followings are forms of expression that produce L-values:

- Subscript expression **e[k]** is an L-value, regardless of whether the expression **e** and **k** are L-values or not.
- A parenthesized expression is an L-value if and only if the contained expression is an L-value. For example, (a)=5 is valid.
- A direct component selection expression **e.name** is an L-value if and only if the expression **e** is an lvalue. For example, mylayer.name is an L-value, but clayey().name is not.

No other form of expression can produce an L-value. Particularly, the result of an assignment expression, the returning of a function call, are never an L-value.

Precedence and Associativity

Each expression operator in TCL has a precedence level and a rule of associativity. An expression may contain a series of operands and operators. The evaluation orders of the expression depends on how these operands are grouped with the operators. Where parentheses do not explicitly indicate the grouping, the grouping of an operand with either of the two operators (left and right) is determined by the rules of precedence and associativity. The operand is grouped with the operator having higher precedence, but if the two operators have the same precedence, the operand is grouped with the left or right operator according to whether the operators are left-associative or right-associative.

For example, in the expression

$$a * b + c$$

the operand *b* is grouped with the multiplication operator '*', because '*' has a higher precedence than '+', and so the expression is treated as if it has been written

$$(a * b) + c$$

Similarly, in the expression

$$a += b != c$$

the operand *b* is grouped with the operator '!=', because '!=' has higher precedence than '+=', and so the expression is treated as if it had been written

$$a += (b != c)$$

But, in the case of the expression

$$a - b + c$$

the two operators '-' and '+' have the same precedence and are left-associative (all operators having the same precedence level always have the same associativity), so the operand *b* is grouped with the operator '-', and the expression is treated as if it had been written

$$(a - b) + c$$

Another case of right-associative operators is

$$a = b += c$$

5L	&&	Logical AND
4L		Logical OR
3R	e1?e2:e3	Conditional
2L	,	Comma
1R	=	Pure assignment, need L-value
1R	*=	Multiplication Assignment, need L-value
1R	/=	Division Assignment, need L-value
1R	%=	Modulus assignment, need L-value
1R	+=	Addition assignment, need L-value
1R	-=	Subtraction assignment, need L-value
1R	<<=	<< Assignment, need L-value
1R	>>=	>> Assignment, need L-value
1R	&=	& Assignment, need L-value
1R	^=	^ Assignment, need L-value
1R	=	Assignment, need L-value

The following operators are not supported in standard C/C++, but is supported in current version of TCL (historical reason). The use of these operators (FORTRAN-like) are not encouraged by the author. They are listed here just in case if your TCL programs accidentally contains one of them (especially '**') in the expression, you may run into troubles that the result isn't as expected while there is no error reported. Some of them may be removed in future release of TCL.

Precedence	Operator	Meaning
16	typeof	Type of operand
16	.NOT.	Logical Negation
14L	**	Arithmetic Power $A^{**}2 = A * A$
14L	.PWR.	Arithmetic Power
13L	.MUL.	Multiplication
13L	.DIV.	Division
13L	.MOD.	Remainder
12L	.ADD.	Addition/string concatenation
12L	.SUB.	Subtraction/string removal
10L	.GT.	Greater than
10L	.GE.	Greater than and Equal to
10L	.LT.	Less than
10L	.LE.	Less than and Equal to
9L	.EQ.	Equal to
9L	<>	Not Equal
9L	.NE.	Not Equal
5L	.AND.	Logical AND
4L	.OR.	Logical OR
3L	.XOR.	Logical XOR
3L	.EQV.	Logical Equivalent
3L	.IMP.	Logical Implication

Difference in Operators

Most of the operations supported by the TCL operators have been extended beyond what the standard C operators provide. Such as, the '+' and '-' can be used with double operand to increment or decrement the value by one, so can the left-shift '<<' and the right-shift '>>' operators to multiply or divide the double operand by a value raised to the power of 2. The most interested and useful operations provided by the TCL operators in addition to those from the standard C's are about the character string and geometry entities, which are the subject of this section.

Complex number arithmetics are also supported by the operators. A complex number can be taken as a point on the X/Y plane, with the X-coordinate as the real part and the Y-coordinate as the imaginary part. So, a point P(X,Y) may mean a complex number expressed as X+Yi.

The following lists these additional operations supported by the TCL operators with respect to different operands. The **ent** means data of geometry entity type like POINT, LINE, ARC and CIRCLE. The **str** means character string. The **value** means a scalar value of integer or double. In the list, the first item at the right, is the return data type, followed by a brief description to the operation.

- ent	nt, Symmetry of ent with respect to (0,0). Valid entity types are POINT, LINE, ARC, CIRCLE. This also includes the negation of a complex number.
str1 + str2	tring, Concatenation result of str1 and str2.
ent + pnt	nt, Translation of ent from (0,0) to pnt. Valid entity types are POINT, LINE, ARC, CIRCLE. This also includes the addition operation between two complex numbers.
pnt + ent	ame as above.
str1 - str2	tring, Result of removing str2 from str1 if str2 is a sub-string of str1.
ent - pnt	nt, Translation of ent from (0,0) to -pnt. Valid entity types are POINT, LINE, ARC, CIRCLE. This also includes the subtraction operation between two complex numbers.
pnt - ent	nt, Translation of -ent from (0,0) to pnt. Valid entity types are POINT, LINE, ARC, CIRCLE.
ent * value	nt, Scaling of entity by the specific value. Valid entity types are POINT, LINE, ARC, CIRCLE.
value * ent	ame as above.
pnt * pnt	nt, Complex number multiplication.
ent / value	nt, Scaling of entity by 1/value. Valid entity types are POINT, LINE, ARC, CIRCLE.
pnt / pnt	nt, Complex number division.
pnt ** value	nt, Power of complex number operation. The pwr() function also accepts complex number.
str1 == str2	ogic value, String comparison.
str1 >= str2	ogic value, String comparison.
str1 <= str2	ogic value, String comparison.
str1 > str2	ogic value, String comparison.
str1 < str2	ogic value, String comparison.
ent1 == ent2	ogic value, Geometry comparison if two entities are equal. Valid entity types are POINT, LINE, ARC, CIRCLE.

Besides the listing above, the '++' and '--' operators also accepts the operand of double type.

Comma Expressions

A comma expression is an expression containing the comma operators. In standard C, the comma is used only to separate the expressions. But, in TCL, if an assignment operator is present, the comma will become an operator. Each separate expression will become operand of the comma operator.

The evaluation order of a comma expression is from left to right. The result of a comma expression in the operand stack will be the result of each operand expression in the stack. There is no popping off the operands as in the standard C. The results of a comma-expression will leave multiple operands on the stack!

For example, in standard C, the expression

```
L-value = exp1, exp2;
```

is legal. Both *exp1* and *exp2* are evaluated, but the execution will assign the evaluated result of *exp1* to the *L-value*, and discard the result of the *exp2*. Yet, in TCL, the interpretation is different. Both the expressions, *exp1* and *exp2*, are evaluated, and both the results are passed to the assignment operator to make an assignment to the L-value. Whether the assignment is valid or not, depends on the data type of the L-value as well as the data type returned by the expressions. For example, the expression:

```
P1 = 10, 20;
```

assigns the POINT P1 with the X and Y coordinates, and is apparently meaningful and valid in TCL. Another example is

```
dist = P1, P2;
```

which calculates the distance between *P1* and *P2*, and then stores the result to the variable *dist*. The reason why TCL deviates from the C language at the comma expressions, lies in the fact that the command line expression evaluator must accept expressions from the user input and must immediately evaluate it. Ease of input becomes a dominant factor in the consideration.

Nevertheless, this should not cause the serious TCL programmers to panic. Because, if the portability of the TCL programs is of major concern, the said expressions may be rewritten in different form:

```
P1 = P(10.,20.);  
dist = V(P1,P2);
```

to achieve the same purpose, while the compatibility with the C language syntax is maintained.

Though, the problem is solved by using the intrinsic geometry functions, while we have a more pleasing way to write the program, there are certain drawbacks that can not be neglected.

As mentioned earlier, if an assignment operator appears in an expression, the commas will be taken as operators rather than expression separators. This will prohibit the use of assignment operators in the cases where the commas must be taken as expression separators. Such cases are expressions in the argument list, and expressions in the **for(*exp1*;*exp2*;*exp3*)** loop control.

The example below shows the problem:

```
func(i=1, j);
```

In standard C, it assigns 1 to *i*, and passes two arguments, *i* and *j*, to *func()*. Whereas in TCL, the comma is taken as an operator, the whole expression is evaluated to be an assignment, which assigns the result of a comma-expression '1,j' to the *i* (the precedence of a comma is higher than the assignment operator). This results in a single value (assuming the assignment is successful), and only one argument is passed!

To make your TCL program compatible with C program and vice versa, the above example should be written as:

```
func(i=I(1, j))
```

to pass only one argument, and

```
func((i=1), j)
```

to pass two arguments, or simply avoid the assignment expression in argument list if more than 2 arguments are to pass.

The same problem also arises in the case of **for** loop control. Most of the C programmers are used to write the program fragment as the example given below:

```
for(i=0, j=0; j<k; i+=2, j++)
```

```
...  
}
```

But this would certainly cause error in TCL programming, since the commas are taken as operators rather than separators. To avoid this, add necessary parentheses to make it clear to TCL. Or, simply avoid this kind of writing style.

Comma Expressions for Assignment

To a simple expression with an assignment operator (`var=exp`), the right side expression to the assignment operator is evaluated first, and then both the L-value and the evaluated result are passed to the assignment operator for the assignment operation, which assigns the result into the L-value. If the data type of the right side operand does not match with the L-value, the assignment operator must do the necessary data type conversion before doing the assignment. If the right side operand can not be converted to match with the data type of the L-value, an error occurs.

Likewise, if the right side expressions are comma expressions (multiple expressions separated by commas), these expressions are evaluated first from left to right in their appearing order. And again, the L-value and these evaluated results are then passed to the assignment operator for the assignment operation. It is apparently that the assignment operator must do some complex conversions before it can have a single data value to assign to the L-value.

In this section, a brief listing of the formats for the complex conversion are given, according to the data type of the L-value.

Scalar Data Conversions

A scalar data means single value like integer, long, and double. The equivalent scalar data conversions can also be invoked explicitly using the `I()` and `V()`. See also `V()` description.

value	his is the simple data type conversion.
pnt	alculate the distance from the origin (0,0) to the given point. The single argument must be of POINT type.
ent	ulate the length of a given entity. Valid types of argument are LINE, ARC, CIRCLE, ELLIPSE and ENTITY. If the argument is an entity handle, the entity it points to will be read in for the length calculation.
pnt,ent	alculate the distance from a given point to another given entity. The first argument must be of POINT type. While for the second argument, valid data types are POINT, LINE, ARC, CIRCLE, and ENTITY.

POINT Data Conversions

The equivalent POINT data conversions can also be invoked explicitly using the `P()`. See also `P()` description.

x,y	Read the point by the X and Y coordinates, where X and Y are scalar values.
rad,(ang)A	Read the point in polar coordinates where rad is the radius value and ang is the angle value in units of degree. Note that the tagged notation <code>()A</code> means the value tagged with an 'A' is an angle value. Should the use of tag character cause any confusion, parenthesis must be used to make it clear. This is not a portable syntax to C/C++ language, but is convenient for operators in command line input.

ent	Read the point from the given entity. Valid argument types are CIRCLE, ARC, POINT, ELLIPSE and ENTITY. For a circle, arc or an ellipse, it is the center point read. For a point argument, it is simply copied. If it is an entity handle, the drawing data pointed by it is read first and then checked for these valid entities.
ent,length	Read the point by measuring along a given entity (the first argument) at a given length (the second argument, double). Valid argument types for the first argument are LINE, ARC, ELLIPSE and ENTITY.
ent,(ang)A	Read a point by measurement based on a given entity (the first argument) at a given angle (the second argument, double, tagged with 'A'). Valid argument types for the first argument are CIRCLE, ARC, ELLIPSE and ENTITY. The angle is in units of degree (must be tagged with 'A'). For a circle, the starting angle is always at absolute zero degree, while for an arc, it is the same as the arc's starting angle.
pnt,ent	Read a point as the one mathematically on a given entity (the second argument) with the shortest distance to a given point (the first argument of type POINT). Valid types of the second argument are LINE, ARC, CIRCLE, and ENTITY.

LINE Data Conversions

The equivalent LINE data conversions can also be invoked explicitly using the L(). See also L() description.

ent	Read the line directly by loading it from an entity handle or another line entity. Valid argument types are LINE, and ENTITY.
ps,pe	Define the line by directly giving two points. The two arguments must be of POINT type. Note that the z-coordinates of the two end points of the line will be assumed as the current entity elevation (@elevation).
ps,pe,z	Define the line by directly giving two points and an elevation value. The first two arguments must be of POINT type, while the third argument, the z-coordinate value for the two end points of the resulted line.
ps,pe,zs,ze	Define a 3D line by explicitly giving two points and two z-coordinate values. The first two arguments must be of POINT type, while the third and fourth arguments, are the z-coordinate values for the two end points of the resulted line, respectively.
ps,dx,dy	Define the line by giving one start point (the first argument of POINT type), and two delta values, each in X and Y directions, respectively, for the ending point. The z-coordinates of the two end points of the resulted line will be assumed as the current entity elevation (@elevation).
ps,len,(ang)A	Define the line by giving one start point (the first argument of POINT type), the length of the line (the second argument of double), and the direction angle of the line (the third argument of double, tagged with 'A') in units of degree. The z-coordinates of the two end points of the resulted line will be assumed as the current entity elevation (@elevation).

ARC Data Conversions

The equivalent ARC data conversions can also be invoked explicitly using the A(). See also A() description.

ent	Read the arc by directly loading it from a given entity handle that points to an arc entity in the drawing database, or from another ARC entity.
------------	--

- elps,ps,pe** Read the ELLIPSE by new start point and ending point specification. Note that the both points need not to be on the ellipse; they are used to calculate the start angle and the ending angle of the ellipse.
- elps,sang,eang** Read the ELLIPSE by new start angle and ending angle specification.

TCL Control Flow

Most of the control flow statements from standard C are implemented in TCL, except the **goto**. In view point of structural programming, the **goto** statement is not necessary for programming. The existence of goto statement can only add the complexity of the program logic, and make the program tracing become a nightmare, like in these old days of FORTRAN IV.

In this chapter, we shall discuss the syntax of these statements of control flow.

Statements

A statement in a program is a meaningful and completed step to be executed. A statement in TCL is

- An expression followed by a semicolon, ';'.
- A blocks of statements enclosed in a pairs of braces ('{' and '}'), which is also called compound-statement.
- A completed control flow statement.

The followings are examples of statements:

```
x = 0;
i++;
printf(...);
{ a+=b; c=0; }
if ( a!=c ) d += 2;
for(j=i=0;i<10;i++) j+=i;
```

Conditional Statements

The general form for a conditional statement is given below:

```
if ( exp1 )
    statement1
else
    statement2
```

If the *exp1* is evaluated to be true (non-zero), then *statement1* is executed; otherwise, if an **else** clause is present, then the statement following the keyword else is executed (*statement2*). Note that the keyword **else** and the statement it introduced is optional to complete the conditional statements. After executing either the *statement1* or *statement2*, the execution continues immediately with the statement following the conditional statement.

This is a two ways conditional statement. To expand it to a multiway conditional statement, simply replace the *statement2* with another conditional statement. Such cascading of conditional statements looks like this:

```
if ( exp1 )
    statement1
else if ( exp2 )
```

```

        statement2
    ...
    else
        statementn
    
```

The '...' means the cascading can be unlimited.

Note that if the *statement1* is also a conditional statement, use the braces to make it a compound statement, to avoid the possible dangling **else** problem. So is the *statement2*, *statement3*, and so forth.

Switch Statement

The switch statement is a multiway branch based on the value of a control expression. A general form is given below:

```

switch(exp)
  case exp1:
    statements;
  case exp2:
    statements;
  ...
  case expn:
    statements;
  default:
    statements
}
    
```

The expression *exp* is evaluated as a control value. Then, the expression after each case is examined and evaluated in their order of appearance. The result is immediately compared with the control value. If the two values are equal, a match is found and the execution continues with the statements immediately after the **case** label, till the end of the **switch** statement. If the control value does not match any cases, and the default label is present, then the **default** label is matched by default. Note that, after control is transferred to a **case** or **default** label, execution will continue through successive statements, ignoring any additional case or default label, until the end of the **switch** statement is reached, or until the control is transferred out of the **switch** statement by a **return**, **break**, or **continue** statement.

The implementation of this **switch** statement is slightly different from the standard C version:

- There is no limitation on what kind of data type the control expression may return, while standard C requires integer type.
- TCL accepts arbitrary expression as the case label, while standard C require an integer constant as the label.
- TCL requires no case label after the default label, while standard C does not have this limitation.

An example to show the use of **switch** statement and the difference from the standard C is given below:

```

switch(left$(date$( ), 3))
  case "Mon":    weekday = 1;  break;
  case "Tue":    weekday = 2;  break;
  case "Wed":    weekday = 3;  break;
  case "Thu":    weekday = 4;  break;
  case "Fri":    weekday = 5;  break;
  case "Sat":    weekday = 6;  break;
  case "Sun":    weekday = 7;  break;
  default:      weekday = 0;   break;
}
    
```

Iterative Statements

There are three kind of iterative statements provided in TCL; namely, they are:

- **while statement** -- which tests an exit condition before each iteration of the statements.
- **do statement** -- which tests an exit condition after each iteration of the statements.
- **for statement** -- which provides a special syntax that is convenient for initializing and updating one or more control variables as well as testing an exit condition.

while Statement

The general form of a **while** statement is given below:

while (*exp*) *statement*

The **while** statement is executed by first evaluating the control expression. If the result is not zero, then the statement is executed. The entire process is then repeated, until the control expression is evaluated to zero, or the control is transferred by the execution of a **return** or **break** statement.

do Statement

The general form of a **do** statement is given below:

do *statement* **while** (*exp*);

The **do** statement is executed by first executing the embedded statement. Then the control expression is evaluated; if the value is not zero, the entire process is then repeated. The execution of the **do** statement is completed when the control expression evaluate to zero or the control is transferred by the execution of a **return** or **break** statement.

The **while** clause may be omitted in the **do** statement in TCL. If so, the embedded statement will be repeated over and over as if the control expression is always true.

For Statement

The general form of a **for** statement is given below:

for ({*exp1*};{*exp2*};{*exp3*}) *statement*

The for statement is executed as below:

1. If the *exp1* is present, it is evaluated and the value is discarded.
2. If the *exp2* is present, it is taken as the control expression and is evaluated. If the result is zero, then the execution of **for** statement is complete. Otherwise, the execution continue to step 3. Note that if the *exp2* is omitted, it is taken as if the evaluation is always true.
3. The body *statement* of the for statement is executed.
4. If the *exp3* is present, it is evaluated and the value is discarded.
5. Go to step 2.

The execution of a **for** statement is terminated when the second expression evaluates to zero, or when the control is transferred outside the for statement by a **return** or **break**

statement. The **continue** statement with the body of the **for** statement has the effect of causing a jump to step 4.

Break Statement

The **break** statement is the keyword **break** followed by a semicolon. Execution of a break statement cause the execution of the closest enclosing **while**, **for**, **do**, or **switch** statement to be terminated. Program control is immediately transferred to the point just beyond the terminated statement.

Continue Statement

The **continue** statement is the keyword **continue** followed by a semicolon. Execution of a continue statement will cause the execution of the body of the closest enclosing **while**, **for**, or **do** statement to be terminated for the next iteration.

Return Statement

The general form of a return statement is given below:

```
return {exp};
```

The execution of a **return** statement will evaluate the optional expression if it is present, and then terminate the current section of function execution and return control (with the evaluated value) back to the point where the function is called. Note that if the function is already the top level function, the return statement will return control back to the system.

TCL Program Structure

The program structure of a TCL program is about the same as that of a standard C program, containing declaration of variables and functions in the source file. However, being a specialized command language to a CAD system, the TCL must incorporate additional rules to support the connections with the system in its program structure, which is the subject of this chapter.

The Main Program

In standard C, the main program of a C program is the function named **main()**, no matter where it is put in the source file. When such a program is executed, the program logic starts from the **main()** function. Arguments from the command line (operating system) can be passed to this entry function.

But in TCL programming, the rules are different. A TCL program may contain multiple entries for program executions, depending on how it is accessed by the operator. If it is invoked by the **RUN** command, it is taken as a single program and a main program entry from the source file must be determined uniquely. If it is accessed by the **CLOAD** command, then multiple entries of program logic will be identified and stored in the system table for later activation from the command line or from other TCL programs in execution.

For a complete TCL program that is to be (or can be) executed by the **RUN** command, the main program is determined uniquely by the first function declaration that appears in the source file. There is no assumption about the class or name of this function. As long as it is the first function body encountered, it is executed as the main program. No arguments will be passed to this entry function.

One of the reasons not to use the **main()**

system commands and runtime functions. A TCL program written for this purpose may have multiple entries of functions declared with the classes of COMMAND or FUNCTION.

Once a TCL program is loaded, all functions of COMMAND class will be registered to the system command table, and functions of FUNCTION class, to the runtime function table. The operator may directly invoke a loaded function of COMMAND class from the command line by entering its name, just like entering **LINE** command to draw lines. The loaded functions can also be referenced from any TCL expressions as if the functions are originally built in the TCL runtime library.

COMMAND Class

A function of COMMAND class in a TCL program will be registered as a system command when it is loaded. The name of the function is then become a system command. As the character cases of a system command is of no matter, the case of the function name is then irrelevant to the system command it represents. However, if such loaded function is to be called from a TCL expression (either from the command line or from a TCL program), the case of the name must match.

When such a loaded function is invoked from the command line as a system command, the original program body (resident in the memory) containing the function will be scanned over again. Memory are allocated for those global variables declared in the program body and the execution is started from that specific function, just like RUNning a TCL program that takes the specific function as the main program.

The only limitation for such loaded function used as command is that it can not be invoked by the use of **command()** function. This is a deliberated limitation to prevent a TCL program from permanently overriding the built-in system commands. In fact, a TCL program may directly call the loaded functions via ordinary function calls (even if they are loaded for use as system commands), so there is no need to use the **command()** function to do the job.

Note that if a loaded function is called via ordinary function call, it is treated as a function of FUNCTION class. All the rules that apply to such class of functions are also applied to it. See next section for details.

FUNCTION Class

A function of FUNCTION class in a TCL program will be registered as a runtime function when it is loaded. The main difference between such a function from that of COMMAND class is that it can only be used as a function in a TCL expression. The case of such functions are always sensitive.

Another difference drawn between these two classes of functions is that the execution of a loaded FUNCTION class function will not cause rescanning over the resident program body containing the function. This means, no global variable declaration of the loaded program body will be effective during the execution of the function. It is therefore prohibited for such a function to reference to a global variables. Consideration in execution speed overhead is the main reason for this limitation.

PART 2

General of TwinCAD System Variables

In this part of the document, all the system variables that can be accessed by TCL expressions are listed in alphabet order. Each of them is prefixed with an '@' sign to notify the use of it within a TCL expression as a reference to it. The '@' sign is in fact not part of the variable name.

Each system variable has been declared with a fixed data attribute. Its data type can be an integer, a double, a POINT or a character string. If one is declared **ReadOnly**, then it is not possible to alter its content by the use of TCL expression. Attempting to do so will incur the error message:

EvalErr #4: Assignment Need L-Value

Most of the system variables will be saved with the drawing file, except those being declared with **Volatile** or **Local** attribute, or those current values are the same as their default values (to reduce the drawing file size). Note that if a system variable is declared with **Local** attribute, it will be saved in a local disk file as a local configuration data.

Some of these variables may be interactively accessed in GUI dialogue window manner by the commands such as **SYSVAR**, **DIMVAR**, **STYLE**, **EMODE** and **DMODE**, and some by individual command prompts. However, this may not be notified individually for each variable in this document.

Note that you may use the **VARLIST** command to generate the listing of all the system variables. And, you may also create user variables by the TCL function **uservar()** and access them in the same way as those system variables.

@\$DISPMODE

Attribute: Integer bit flag, Read Only, Default=0

Description: The content of this variable stores the current drawing mode of graphic display control. This is a read only variable to TCL program. Useful bit flag informations are given below:

Bit 0-6: Reserved.

Bit 7: 1, if system in 3D view, 0, if in 2D plane view.

Bit 8-15: Reserved.

@\$VIEWX

Attribute: Double, Read only, Default=0.

Description: The content of this variable stores the X-axis component of the current 3D view point, used in **VPOINT** command, valid only when the system is in 3D-view.

@\$VIEWY

Attribute: Double, Read only, Default=0.

Description: The content of this variable stores the Y-axis component of the current 3D view point, used in **VPOINT** command, valid only when the system is in 3D-view.

@\$VIEWZ

Attribute: Double, Read only, Default=1.

Description: The content of this variable stores the Z-axis component of the current 3D view point, used in **VPOINT** command, valid only when the system is in 3D-view.

@ACADTADJ

Attribute: Integer, On-off control, Default=0 (OFF)

Description: The content of this variable specifies how the text justification calculation will be done for all the texts in generation. If it is ON, the system will use the AutoCAD's method to do the text justification calculation, which uses the actual text stroke extent for the center or right justification control. If it is OFF, which is the default, the system will do the calculation, using the text stroke bounding box, in its native way.

The reason to provide this additional control is to make the drawing that contains texts with right/center justification control loaded from a DWG file to appear the same as it was in AutoCAD, especially when the texts are drawn to align within some frames.

The native text justification calculation provided by the system is different from that provided by the AutoCAD. The system always uses the text bounding box (which may also include the left/right side-baring values) for the justification calculation, while it is the actual text stroke extent used by AutoCAD for the same calculation.

There are certain drawbacks to use the text stroke extent for the text justification calculation. For example, if you choose to right-justify two text strings using fix-pitch font (all Chinese fonts are fix-pitch fonts by natural) with the same vertical alignment, the text characters may not align vertically if the last characters on both strings are not the same. The space character, which draws no strokes, is also a problem.

However, if compatible alignment of text is a problem in loading and plotting DWG files, you may set this variable to ON, so that the system will use the AutoCAD's method to align the texts. In fact, the system will automatically set this variable to ON state, if a DWG file is **NEW/Loaded** and it was not saved by the system previously.

@AJOINEPS

Attribute: Double, Local, Default=0.00001

Description: The content of this variable specifies the epsilon value that determines whether two points coincide in the **AUTOJOIN** and **PJOIN** command operations. This variable is also honored by some other command operations that need to determine whether two points coincide in a practical sense. Note that you should never set this value larger than the default value, unless you know what you are doing for.

@ALLTRNSCMD

Attribute: Integer, On-off control, Volatile, Default=OFF (0)

Description: The content of this variable specifies whether or not to accept all commands in transparent mode. This variable can only be accessed via TCL expressions. The default is OFF, i.e., only a predefined set of commands can be issued in transparent mode.

The use of this variable should be restricted in current release for safety reason. The users should be prohibited from turning this variable ON. It is provided for special applications that involves TCL programming, such as in the **WIRECUT** command (which turns this variable ON automatically).

@ANSWERMODE

Attribute: Integer, On-off control, Local, Default=ON (1)

Description: The content of this variable specifies whether or not to display the evaluated result of an expression input from the main command line or input as a transparent command.

@APERTURE

Attribute: Integer, Local, default=6

Description: The content of this variable specifies the size of the target box used in object snapping, in units of pixel. It can be directly accessed by the **APERTURE** command.

@ARCSEGANG

Attribute: Double, angle value in units of radians, default=10°

Description: The content of this variable specifies the increment angle value used to generate a circular segment for the thickness display in a 3D view. Note that the value must be stored in units of radians, though the display of it is in units of degree from the **SYSVAR** command.

This variable also specifies the fineness of an ellipse or elliptic arc or a circular arc with wide-line property drawn in the graphic display.

@ASKFORTAG

Attribute: Integer, On-off control, Default=ON (1)

Description: The content of this variable specifies whether the system will automatically ask the user to apply tags to the new block definition at **BLOCK** command or to the symbol in creation in **SYMLIB** command, when there are tag groups available. If it is ON and there are tag groups available in the drawing, the system will ask the user whether to apply tags or not. If it is OFF, the query will be bypassed and no tags will be applied automatically.

@AUTOAUDIT

Attribute: Integer, On-off control, Local, Default=OFF (0)

Description: The content of this variable controls the auto-audition function. If it is ON, all newly loaded drawings (by **LOAD**, **INSERT**, **NEW**, **DWGIN** commands, etc.) will be audited automatically. The default is OFF.

Note that auditing a drawing may take a little while.

@AUTOQTEXT

Attribute: Double, Ranged from 0 to 1, Local, Default=0.02

Description: The content of this variable specifies the threshold control value, which is the minimum ratio of the text size to the height of the current drawing window, used in the auto-switching between quick-text box display and real vector generation of the text entities in display.

When the quick text mode is OFF, all the text entities will be drawn exactly what they are in the drawing window, even if the text is relatively very small in size to recognize. It takes time to generate these text vectors if the drawing contains a lot of texts of fancy styles, especially when the drawing window is zoomed to view the whole large drawing.

To solve this problem, a quick text mode threshold control is provided, which causes the display of a text entity in the graphic window as a single box when the ratio of the text size in height to the height of the drawing window is smaller than this threshold value. And thus, these small text in the drawing window will be drawn far more quickly than ever as boxes, while you still can read the other texts that are large enough to recognize from the window. To view these small texts, just to zoom the drawing so that they become large enough to display.

To effectively disable this threshold control function, set this variable to zero. It is effective only when the quick text mode is off. Note that you may turn the quick text mode ON to make the drawing display faster; however, this will make all the text entities be displayed as boxes. Once you want to read the text, you will have to turn it off and regenerate the drawing again.

@AUTOREDRAW

Attribute: Integer, On-off control, Local, Default=ON (1)

Description: The content of this variable controls the auto screen refresh in the midst of a prime command execution. This is an internal variable and may be accessed by TCL expression only. It is effective only when the prime command is issued from **command()** and is completed totally by the script (at least to the point where the screen refresh is encountered).

Certain Prime Commands will issue screen refresh request to the Graphic Runtime to refresh the screen, so that these highlighted objects (on temporary basis) or pointer marks (if **BLIPMODE** is ON) will be removed

from the display. The refreshing will cause the Graphic Runtime to clear the graphic window and redraw the display list again. Usually, this is fast and latent to the operator. Even for large drawing, the redrawing time is still tolerable to the operator in terms of man/machine interaction.

However, for a TCL application that must issues successive **command()** calls to modify objects, the accumulated time delay wasted on the screen refresh may be significant. So, to maximize the performance of such a TCL application, setting this variable to 0 shall be very helpful. Of course, the TCL application must issue the **redraw()** to refresh the screen by itself at the appropriate time.

@AUTOREGION

Attribute: Integer, on-off control, Volatile, Default=1 (ON)

Description: The content of this variable specifies whether the **HATCH** command will create a region entity to associate with the user defined hatch pattern given by solid hatching lines. If it is ON, a region entity will be created to cover the hatch area, and the hatching lines specified by the user will become an associated hatching specification with that region. If it is OFF, the **HATCH** command creates a drawing block of the real hatching lines.

Note that the **HATCH** command will always create a region entity for the hatching lines generation in an associative manner if a predefined hatch pattern is used.

It is strongly recommended to leave this variable ON and let the **HATCH** command create associatively hatched regions for the user defined hatching lines. There are many advantages in using associatively hatched regions over the old hatching blocks. For example, capable of direct changing the hatching pattern via the **CHANGE** command is one of the many. And, You may, however, use **EXPLODE** command to convert an associative hatched region into an equivalent hatching block when it is necessary.

Note: This variable's attribute was altered from Local to Volatile since Version 3, Release 3. All volatile variables are always initialized to their default values at system start-up.

@AUTOSAVE

Attribute: Integer, Number control, Local, Default=0

Description: The content of this variable controls the **AUTOSAVE** feature by specifying the autosaving interval in units of effective command numbers. If it is zero, the **AUTOSAVE** feature is disabled. All else values will enable the feature to save the drawing at every specific number of effective commands.

An effective command is a command that affects the drawing database and that can be undone. Note that this will exclude the **UNDO/REDO** commands.

You may also specify to save the drawing automatically by time, via the system variable **@SAVETIME**.

@AXOSCALE

Attribute: Double, Default=1.

Description: The content of this variable controls the Auto-scaling function in PUCS plane setup (done by **PUCS**, **AXOPLANE** and **ISOPLANE** commands).

If it is zero, the Auto-scaling function in the projection plane setup will be disabled. When a set of projected axes is constructed by the use of the **PUCS**, **AXOPLANE** or **ISOPLANE** command, the unit vector length of each of the projected axes will be set to its natural foreshortening factor on that axis under the Axonometric Projection condition and the value of the **@MVSCALE** will always be reset to zero.

If it is not zero, then the Auto-scaling function in the PUCS plane setup will be enabled for the **PUCS**, **AXOPLANE** and **ISOPLANE** commands to setup the unit vector lengths of the major projected axes (for Isometric or Dimetric Projection) to follow the drawing conventions. A fixed scale factor is applied over the natural foreshortening factors for the Isometric or Dimetric Projection axes setup. The same scale factor will be applied to the subsequent axes setup for skew planes under the same projection view. A skew plane projection axes setup can be obtained by **PUCS/Rotate** or by **AXOPLANE/Trimetric** commands.

@BEVANGLE

Attribute: Double, Angle value, Default=0°.

Description: The content of this variable controls the beveling of jointed wide lines (wide polyline segments).

If its value is zero, it specifies a default beveling, which starts only when the corner angle is less than 90°. The beveling point is fixed and equal to that set by the angle value of 90°.

If its value is positive, it specifies the corner angle below which to start the beveling. The beveling point will be determined by this angle value (it will be a point extended out from the center line end point in half of the setting angle direction).

If its value is negative, its absolute value also specifies the corner angle below which to start the beveling. However, the beveling point is fixed at the original wide line outer end corner.

@BLIPMODE

Attribute: Integer, On-off control, Default=On (1)

Description: The content of this variable controls the blip point marking function, which generates a point marker (in small crossline) right at the snapped position when you pick up an object or designate a point. The marker thus generated is not part of the drawing, yet helps you to identify the pickup operation. It will be removed when you redraw the screen.

This variable is accessed by the **BLIPMODE** and **DMODE** command.

@BLOCKCTRL

Attribute: Integer, Bit-flag, Local, Default=0

Description: The content of this variable control the details of block related operations. Currently supported bit-flags are described below:

- | | |
|-------------|---|
| 0 | 1, Ignore all duplicated blocks during drawing loading by LOAD , DWGIN and INSERT command. |
| Else | Reserved, and should be all zeros. |

@BPOLYLEVEL

Attribute: Integer, Local, Default=255

Description: The content of this variable stores the current setting of the maximum area nesting level value for the **BPOLY** operation.

A close area not contained in any other close areas has an area nesting level of zero. The area bounded by a close contour within an area of level zero has an area nesting level of one, which can be taken as a hole to its containing area. Likewise, the area bounded by a close contour within an area of level one, has an area nesting level of 2. Since the area of level 1 is a hole, the area of level 2 is then an island to the hole. And so on, till all the areas are assigned with a proper nesting level value.

The maximum area nesting level setting value is then used to control the depth of the nesting level that the **BPOLY** operation will use to recognize those areas under **@BPOLYMODE** (ON). If you set this value to zero, the **BPOLY** operation will take only level zero areas into account and ignore all the holes and islands that may be found within. If it is set to 1, then holes of level 1 are accepted but not islands. Likewise, if it is set to *N*, then all the areas of which the nesting levels are larger than *N* will be ignored.

@BPOLYMODE

Attribute: Integer, On-Off control, Local, Default=1 (ON)

Description: The content of this variable specifies whether the **BPOLY** command will take the selected loops (close polylines) for a regional primitive or not.

If it is ON, the **BPOLY** command will create a solid-fill Region entity from those selected loops. During the **BPOLY** command operation, all the selected loops will be highlighted with solid-fill color (to reflect that it is the area interested), and their total regional area as well as their overall geometry center will be calculated and displayed automatically using the even-odd rule under the current setting of **@BPOLYLEVEL**.

However, if it is OFF, all the selected loops during **BPOLY** command processing will not be taken as regional primitives (that contributes to a close area) but rather only close contours independent of each other. And thus, they will be highlighted with dashed lines only around the coutou-21N-oddstnLY

off. However, if it is totally on the indicated side and is not crossed by the boundary object, it will be removed.

@BYBLKCOLOR

Attribute: Integer, Color value, Local, Default=-1 ([ByLayer])

Description: The content of this variable specifies the actual color number used for the [ByBlock] color on a non-block-element entity. The system default is to use the [ByLayer] color.

To create a block containing entities with [ByBlock] color, the user must create those entities with [ByBlock] color first. However, before those entities become the constituents of a block, they must exist by themselves. The [ByBlock] color is thus must be defined in such circumstance.

The earlier version of **TwinCAD** always use [ByLayer] color for such entities. However, the color used for such entities in AutoCAD is never a [ByLayer] color, though it may be configurable to a fixed color.

@CDSPFILE Dos

Attribute: String, Filename, Volatile, Readonly, Default=NULL

Description: The content of this variable stores the current active DSP file used by the system. This active DSP file is setup by the **.DSPFILE** command. A DSP file is a message display file that is designed to work with an INP file and used in the *Playback/Recording Function*.

@CHAMDIST1

Attribute: Double, Default=0

Description: The content of this variable specifies the first chamfer distance used in **CHAMFER** command.

@CHAMDIST2

Attribute: Double, Default=0

Description: The content of this variable specifies the second chamfer distance used in **CHAMFER** command.

@CHELPPFILE

Attribute: String, Filename, Volatile, Default=Depending on initialization.

Description: The content of this variable specifies the filename of system help file, which is read whenever the user enter the **'HELP** command.

@CMDECHO

Attribute: Integer, On-off control, Volatile, Default=ON (1)

Description: The content of this variable controls whether to echo the command input when the input is from the menu, script, or TCL programming. If there is no TCL program in execution, when the command echo is turned off, then only those input from the menu or scripts be suppressed from the display.

However, if the command is invoked by a TCL program, most of the system prompting messages will also be suppressed.

Note that the On/Off state of this variable can be toggled by the direct control command **<Ctrl/^\>**, code **1EH**.

@CMENUFILE

Attribute: String, Read only, Volatile, Default=Depending on initialization.

Description: The content of this variable stores the filename of current loaded menu file. Use **MENU** command to load menu file.

@COLORMASK

Attribute: Integer, Bit-flag control, Volatile, Default=0

Description: The content of this variable specifies the entity selection mask by entity's color. The 16 color numbers corresponds to the 16 bits of this integer value from bit 0 (LSB) to bit 15 (MSB). If a bit is set to 1, the entity with the corresponding color number will be rejected in the object selection operation, provided that the selection masking is also enabled. The **BYLAYER** and **BYBLOCK** colors of an entity are always resolved to the actual color number used for it before doing the entity sieving by the mask.

@CSYMLIB

Attribute: String, Read only, Default=None

Description: The content of this variable stores the filename of current active symbol library, accessed by **SYMBOL** and **SYMLIB** command.

@CURCOLOR

Attribute: Integer, Ranged from -1 to 255, Default=0

Description: The content of this variable specifies the color number for subsequent new entity creation except dimension entities. It is called the current entity color. A **-1** value means the entity color is **BYLAYER** and a **-2** value, **BYBLOCK**. Supported color values are from **0** to **15** in current version. Values falling off this range will be accepted, but only the lower four bits value are effective for the display color.

Note that the effective display color for **BYBLOCK** color is specified by the system variable **@BYBLKCOLOR**.

@CURLAYER

Attribute: Integer, Default=0

Description: The content of this variable specifies the current layer by its ID number, which ranges from **0** to **255**. Use **layerid()** to obtain the ID number of a layer. Note that the layer **'0'** is always assigned with the ID number **0**.

@CURLTYPE

Attribute: Integer, Default=0

Description: The content of this variable specifies the current linetype ID number for the subsequent entity creation. A **0** means the linetype is **BYLAYER**. Use *ltypeid()* to obtain a linetype's ID number. Note that the linetype '**CONTINUOUS**' is always assigned with the ID number **1**.

@CWORKFILE

Attribute: String, Volatile, Default=Depending on initialization.

Description: The content of this variable stores the filename of current work drawing file.

@DIMADPRE

Attribute: Integer, Default=-1

Description: The content of this variable specifies the number of decimal precision for the default text generation of angular dimension. The effect of this variable setting is the same as that of **DIMDPRE** to other type of dimensions, except the following:

- 10 Output degree only, as ddd~°~
- 11 Output degree and minutes, as ddd~°~ mm'
- 12 Output degree, minutes and seconds, as ddd~°~ mm'ss"

See also **DIMDPRE** for the effects produced by other value settings.

@DIMAOPT

Attribute: Integer, Bit flag control, Default=0

Description: The content of this variable controls the default feature options for the angular dimensioning. Useful bit flag informations are given below:

- Bit 0** Dimension Line Option (OD), specifying whether to add a circular arc across the angle to measure when the dimension lines (with arrows) are outside of the angular dimension. If this bit flag is ON, the additional arc will be added joining the two dimension lines at the arrow tips.
- Bit 1** Text Generation Option (OT), specifying whether to turn the circular text generation off or not. If this bit flag is ON, circular text generation will be turned off, and the text generated for the angular dimension will always be horizontal. If this bit flag is OFF, the text will be in circular fashion around the angle vertex, provided that the OA bit is also OFF.
- Bit 2** Text Alignment Option (OA), specifying whether to align the dimension text with the line drawn from the angle vertex to the text position, in such a manner that the Datum Dimensioning can be achieved. This bit flag is effective only when the OT bit is OFF.
- else** Reserved, should be zero for future compatible mode.

These above default feature options are assigned to the new angular dimensions in subsequent creation. The interface control of **OD** and **OT** options are also provided when you are dragging the angular dimension (creation by **ADIM** or modification by **CHANGE** command). See also **ADIM** command in Command Reference.

@DIMARND

Attribute: Double, Default=0.

Description: The content of this variable specifies the round-off unit for the angular dimension measurement in the default text generation. If it is zero, the round-off function is disabled. For example, to make the default text generation of the angular dimension in units of 5°, you may specify a value of 5 to this variable.

@DIMASIZE

Attribute: Double, Default=N/A

Description: The content of this variable specifies the size of the arrow pointer drawn at the end of a dimension line. This is a global dimension variable. Changing this variable will change the size of all dimensioning arrow pointers after drawing regeneration.

@DIMASTR

Attribute: String, At most 16 character, Default="~陡"

Description: The content of this variable stores the unit suffix following the angular dimension measurement in the default text generation. Generally, the degree (°) symbol should be used for the suffix; however, depending on the text style in use, you have to enter the correct code that represents the symbol. Note that, to avoid the code conflict with the extended character style, use the '~' function to isolate the special symbols.

@DIMATYPE

Attribute: Integer, Default=0

Description: The content of this variable specifies the type of arrow pointer used for the next subsequent creation of dimension. This value will be saved with the dimension entities. Currently, there are 16 kinds of arrow pointers supported in this version, from value 0 to value 15. See **DIMVAR** command for details.

@DIMCEN

Attribute: Double, Default=0.

Description: The content of this variable controls the drawing of Circle/Arc center marks and center line by the **DDIM** and **RDIM** commands. If it is greater than zero, it specifies the size of the center mark, and the center mark is drawn as well. If it is less than zero, the center line is drawn besides of the center mark; if it is zero, both are not drawn.

It is recommended to use the **CENDIM** command to create the center lines/marks independently.

@DIMCOLOR

Attribute: Integer, color number, Default=N/A

Description: The content of this variable specifies the color used for subsequent creation of dimension entities. This value will be saved with each dimension entity.

Note that this variable overrides the **@CURCOLOR** in the creation of dimension entities. A -1 means **BYLAYER**.

@DIMDBSTR

Attribute: String, Maximum 16 characters, Default="~φ~"

Description: The content of this variable specifies the prefix added before the diameter dimension measurement in the default text generation. Generally, the DIA (φ) symbol is used for the suffix; however, depending on the text style in use, you have to enter the correct code that represents the symbol.

To avoid the code conflict with the extended character style, you may use the '~' function to isolate the special symbols.

@DIMDESTR

Attribute: String, Maximum 16 characters, Default=""

Description: The content of this variable specifies the suffix following the diameter dimension measurement and the **DIMPOST** in the default text generation. It is an alternative counterpart of the **DIMDBSTR**.

@DIMDLIR

Attribute: Double, Default=2.

Description: The content of this variable specifies the default Dimension Line Increment Ratio, which is used to calculate the default increment value of dimension line position based on the height of dimension text. The actual default dimension line increment will be the height of the dimension text multiplied by the dimension scale factor and by this ratio value.

For example, if the dimension text size is 5, and a scale factor of 1 is used, with the default **DIMDLIR** value of 2, the actual default dimension line increment will be $5 \times 1 \times 2 = 10$ drawing units.

@DIMDOPT

Attribute: Integer, Bit flag control, Default=0

Description: The content of this variable controls the default feature options for diametric dimensioning. Valid bit flag informations are given below:

Bit 0 Dimension Line option (OD), specifying whether to generate the dimension line (with arrow) inside of the circle while the dimension text is outside of it. It affects only when the dimension text is placed outside of the circle to dimension. If this bit flag is OFF, the dimension line and arrow will be generated outside pointing to the circle. If it is ON, the dimension line will extend across the diameter of the circle and the arrow pointers are generated as if the dimension text is placed inside of the circle.

Bit 1 Text Generation Option (OT), specifying whether to align the dimension text with the dimension line or not, effective only when the dimension text is placed outside of the circle. If this bit flag is OFF, the dimension text generated will always be horizontal. But if it is ON, the dimension text will be generated in the direction as the dimension line.

Bit 2 Text Alignment Option (OA), specifying whether to align the dimension text above the dimension line or not, effective only when the text is placed outside of the circle. If it is OFF, the dimension text will be placed at a position such that the dimension line leads to the middle height of the text. If it is ON, the text will be placed above the dimension line (dimension line will be extended to cover the span of text).

Note that this feature option affects only the automatic placement of text when it is in dragging for creation or modification. It is used only to specify the default text position of a dimension entity when it is created or modified interactively. In fact, you may use SETDIM to move the dimension text to other position.

Bit 3 Dimension Line Reverse Option (OR), specifying whether to reverse the dimension arrow pointer direction when the OD option is ON, effective only when OD is ON. If this bit flag is ON, and the OD option is also ON, then the arrow pointers that were generated inside of the circle will be generated outside pointing into the circle. The dimension line will also be extended over the circle for the additional arrow pointer outside of it.

else Reserved, should be zero for future compatible mode.

These above default feature options are assigned to the new diametric dimensions in subsequent creation. The interface control of **OD**, **OT**, **OA** and **OR** options are also provided when you are dragging the diameter dimension (creation by **DDIM** or modification by **CHANGE** command). See also **DDIM** command in Command Reference.

@DIMDPRE

Attribute: Integer, Default=-1

Description: The content of this variable specifies the number of decimal precision to output in the default text generation for dimension measurement. If a zero is specified, then the default text generation will contain only the integer part of the measurement. The decimal point will also be removed. If a positive number is specified, then the default text will contain the specified number of digits after the decimal point. If a negative number is specified, the absolute value of the number is also used to specify the number of decimal precision after the decimal point, but the trailing zero of the resulting text string will be removed.

One exception is the number -1, which will take 3 decimal precision digits in the default text generation without the trailing zero, for compatible reason (to the very earlier version).

@DIMDRLEAD

Attribute: Double, Default=0.

Description: The content of this variable specifies the default length of the horizontal lead line to the text for the DDIM and RDIM command when the text is dragged outside of the dimensioned circle. If the value is positive and non-zero, it specifies the absolute length value of the lead line. If the value is negative, it specifies the ratio value to the current arrow size in use. If it is zero, which is default and compatible to the previous setting, it is equivalent to -1, which specifies to use the current arrow size as the default length.

Note that this variable affects only the user interface in the creation and

Description: The content of this variable controls the default feature options for linear dimensioning. Valid bit flag informations are given below:

- Bit 0** Dimension Line Option (OD), specifying whether to add a line across the dimensioning when the dimension lines (with arrows) are outside of the angular dimension. If this bit flag is ON, the additional line will be added joining the two dimension lines at the arrow tips.
- Bit 1** Text Generation Option (OT), specifying whether to align the text angle with the dimension line or not. If this option is ON, the default dimension text angle will be zero degree (always in horizontal orientation). And, if the dimension text is outside of the dimension line, additional lead line will be drawn to the text (depends on where the text is placed and the status of the OA option). If this option is OFF, the default dimension text angle will be determined by the orientation of the dimension line. This is a new features in V3.0, and is not downward compatible.
- Bit 2** Text Alignment Option (OA), specifying the way to align the dimension text with the dimension line. If this option is ON, the dimension text will be HC-justified and aligned with the dimension line. If the text is placed inside the dimension line, the dimension line will be clipped. If this option is OFF, the dimension text will be placed above the dimension line (left or center justified depends on @DIMITADJ) by a specified gap distance. This is a new features in V3.0, and is not downward compatible.
- Bit 7** Aligned Measurement Option. This is a global control bit flag affecting the dimension value of a rotated linear dimension only when it is being created or modified by CHANGE command. It controls the determination of dimension value of a rotated ALDIM.
- If this bit flag is ON, then the measurement value will be taken from the true distance between the two dimension points in despite of the Rotated measurement request. If this bit flag is OFF, which is the default, the measurement value will be the projected value of the distance in the rotated direction.
- Bit 8** Spline Leader Option for Leader Dimension. 1 if for spline leader, and 0 for linear leader.
- else** Reserved, should be zero.

These above default feature options will be assigned to the new dimensions in subsequent creation. The interface control of **OD** option is also provided when you are dragging the linear dimension (in creation by **HDIM**, **VDIM**, **ALDIM** and **LDIM**, or in modification by **CHANGE** command).

@DIMLTOL

Attribute: Double, Default=0

Description: The content of this variable specifies the lower tolerance of the dimension in the subsequent creation of dimensions. This tolerance value will be negated before use in the text generation. Note that if both the **DIMUTOL** and **DIMLTOL** are equal, the tolerance will be expressed by using the plus-minus sign.

@DIMOBLANG

Attribute: Double, angle value in units of radians, Default=0.

Description: The content of this variable specifies the oblique angle used in the dimension text generation. This is a global and dynamic variable. Changing this variable will affect all the existing dimension texts. Note that the oblique angle of from the Oblique Dimensioning (from **ALDIM**) will override this value.

@DIMOBLARW

Attribute: Integer, On-off control, Default=OFF (0)

Description: The content of this variable controls the oblique feature of arrow pointers used in dimensioning. This is a global and dynamic variable, default to OFF for compatible reason.

This oblique feature, if turned ON, will generate the oblique arrow pointers in linear dimensions with oblique specification. This feature will make the open end of an arrow pointer on an oblique dimension be aligned with the extension line at the arrow sharp end and thus appeared with a projecting effect.

@DIMPFIX

Attribute: String, Default=""

Description: The content of this variable specifies a default prefix string to the linear dimensioning.

@DIMPOST

Attribute: String, Maximum 22 characters, Default="mm"

Description: The content of this variable specifies general unit suffix following the dimension measurement in the default text generation. It applies to all types of dimensions except the angular dimension.

@DIMRBSTR

Attribute: String, Maximum 16 characters, Default="R"

Description: The content of this variable specifies the prefix added before the radius dimension measurement in the default text generation.

@DIMRESTR

Attribute: String, Maximum 16 characters, Default=""

Description: The content of this variable specifies the suffix following the radius dimension measurement and the **DIMPOST** in the default text generation. It is an alternative counterpart of the **DIMRBSTR**.

@DIMRND

Attribute: Double, Default=0.

Description: The content of this variable specifies the rounding unit for the dimension measurement in the default text generation. If it is zero, the function is disabled. For example, suppose the rounding unit is 5, then the measurement of a dimension will be rounded off to a multiple of 5.

@DIMROPT

Attribute: Integer, Bit flag control, Default=0

Description: The content of this variable controls the default feature options for radius dimensioning. Valid bit flag informations are given as below:

Bit 0 Dimension Line option (OD), specifying whether to extend the dimension line (with arrow) across the diameter into the circle while the dimension text is outside of it. It affects only when the dimension text is placed outside of the circle to dimension.

If this bit flag is OFF, the dimension line and arrow will be generated outside pointing to the circle. If it is ON, the dimension line will extend across the diameter into the circle and the arrow pointers are generated at the end of the dimension line pointing to the circle. The effect of OD will be affected by OR. See also OR feature.

This feature option is for dimensioning an arc of small radius, when the text is to be placed "inside" of the arc. See RDIM command for examples.

Bit 1 Text Generation Option (OT), specifying whether to align the dimension text with the dimension line or not, effective only when the dimension text is placed outside of the circle. If this bit flag is OFF, the dimension text generated will always be horizontal. But if it is ON, the dimension text will be generated in the direction as the dimension line.

Bit 2 Text Alignment Option (OA), specifying whether to align the dimension text above the dimension line or not, effective only when the text is placed outside of the circle. If it is OFF, the dimension text will be placed at a position such that the dimension line leads to the middle height of the text. If it is ON, the text will be placed above the dimension line (dimension line will be extended to cover the span of text).

Note that this feature option affects only the automatic placement of text when it is in dragging for creation or modification. It is used only to specify the default text position of a dimension entity when it is created or modified interactively. So, you may use SETDIM to move the dimension text to other position.

Bit 3 Dimension Line Reverse Option (OR), specifying whether to reverse the dimension arrow pointer direction when the OD option is ON, or to extend the dimension line to the center of the circle if the OD option is OFF, effective only when the dimension text is placed outside of the circle.

If this bit flag is ON, the dimension line will be extended to the center of the circle, and if the OD option is ON, then the arrow pointer will be inside of the circle, and if the OD option is OFF, the arrow pointer will be outside. See RDIM command for examples.

else Reserved, should be zero.

These above default feature options are assigned to the radius dimensions in subsequent new creation. The interface control of **OD**, **OT**, **OA** and **OR** options are also provided when you are dragging the radius dimensioning (creation by **RDIM** or modification by the **CHANGE** command). See also **RDIM** command.

@DIMSCALE

Attribute: Double, Default=0.

Description: The content of this variable specifies the scaling factor applied to the size of dimension text and arrow for subsequent dimension entity creation. If it is zero, the scaling is disabled (equivalent to a scale factor of 1). As the scale factor will be converted into integer (by a scale factor of 256) stored with the dimension entity, the maximum acceptable scale value is therefore limited to 255.99 with a resolution of 1/256, which is quite sufficient for most of the cases.

@DIMSE1

Attribute: Integer, On-Off Control, Default=OFF

Description: The content of this variable specifies whether to suppress the first extension line of the dimensions created subsequently. If it is ON, the first extension line will be suppressed. It is effective only to linear and angular dimensionings.

@DIMSE2

Attribute: Integer, On-Off Control, Default=OFF

Description: The content of this variable specifies whether to suppress the second extension line of the dimensions created subsequently. If it is ON, the second extension line will be suppressed. It is effective only to linear and angular dimensionings.

@DIMTADJ

Attribute: Integer, Selection among 0, 1, and 2, Default=0

Description: The content of this variable specifies the text adjusting type.

A value of **0** specifies that the dimension text is dragged by its start position, i.e., the text is left-adjusted. This is the default condition.

A value of **1** specifies that the dimension text is dragged by its middle position, i.e., the text is center-adjusted. It is effective only to linear and angular dimensionings.

A value of **2** specifies that the dimension text will be automatically placed at the center of the dimension line, whenever the dragging of the text is inside of the dimension. Note that it is not effective to the ordinate dimensions.

All else values are ignored and assumed as the value 0.

@DIMTCDIST

Attribute: Double, Default=0.

Description: The content of this variable specifies the text character distance applied dynamically to the dimension texts in generation. It specifies the character distance or space between characters in the same way as the system variable @TXTCSPACE does for the ordinary Text Entities. See also @TXTCSPACE.

@DIMTFAC

Attribute: Double, Default=1.

Description: The content of this variable specifies the text height scale factor used in subscript and superscript generation for the dimension texts. The subscript and superscript are used for tolerance expression.

The base text height of subscript or superscript is half of the normal text height. You may change this text height by setting an appropriate scale value to this variable. For example, to use full height of text in the tolerance expression, you may set this scale factor to 2 or -2.

The baseline alignment of the superscript and subscript texts can be controlled by the sign of this variable. If the setting value is positive, the text alignment will force the bottom line of the superscript be located at the half-height position to the normal text. If it is negative, the text alignment will use the original baseline for the subscript text and calculate the baseline position of the superscript.

@DIMTOL

Attribute: Integer, Default=0

Description: The content of this variable controls the type of tolerance generation and specifies the number of decimal positions the tolerance will be expressed with.

If it is positive, it enables the tolerance expression in default dimension text generation. If it is zero, then the tolerance expression generation in dimension creation will be disabled. If it is negative, then the default text will generate the upper limit and lower limit for the subsequent dimensionings.

@DIMTRDIST

Attribute: Double, Default=0.

Description: The content of this variable specifies the default text row distance used for the dimension text generation, such that the specification of **TXTRDIST** will not affect the dimension text generation, especially when there are tolerances or limits dimensioning.

@DIMTSIZE

Attribute: Double, Default=N/A

Description: The content of this variable specifies the height of the dimension text in use. This is a global and dynamic variable. Changing this variable will change the size of all dimensioning texts, excepts those being exploded, after the drawing regeneration.

@DIMTSTYLE

Attribute: String, Style name, Default="STANDARD"

Description: The content of this variable specifies the text style to use for the dimension texts. This is a global and dynamic variable. Changing this variable will change the shape of all dimension texts, excepts those being exploded, after the drawing regeneration.

@DIMTWIDTH

Attribute: Double, Default=1.

Description: The content of this variable specifies the text width factor used in the dimension text generation. This is a global and dynamic variable. Changing it will affect all the existing dimension texts.

@DIMUTOL

Attribute: Double, Default=0.

Description: The content of this variable specifies the upper tolerance of the dimensioning for the subsequent creation of dimensions.

@DISPCTRL

Attribute: Integer, Bit-flag, Local, Default=0.

Description: The content of this variable provides some special system controls. Effective bit flag value are described below:

Bit 0 Specifies whether the logical coordinate space for the display list will be made equal to the physical window's dot space or not. If it is ON, these two coordinate spaces will be made equal, and some overhead in address mapping calculations may be reduced (depends on the version of the Graphic Runtime loaded) so that the graphic speed can be faster (yet with less precision). If it is OFF, a homogeneous fixed logical coordinate space (device independent) will be used, which is a compatible mode for old INP file to work.

This bit flag also affect the generation of display list for the PRPLOT command.

Bit 1 Specifies whether to expand an ARC command vector into LINE vectors before storing it into the display list or not. If it is ON, the Graphic Runtime (also depends on the version) may expand an ARC command vector into successive LINE vectors before storing it. This will speed up the REDRAW (screen refresh) command as well as the PRPLOT processing, however, in the expanse of consuming more display list memory. If it is OFF, the original ARC command vector will be stored directly in the display list.

Else Reserved and should be zero.

@DISP_POSNO

Attribute: Integer, Local, Default=0.

Description: The content of this variable specifies the decimal position of all coordinate, length, angle values in display with the output conversion format specifier "%@0&f". It also affects the screen coordinate display.

If it is zero, the output conversion done by the "%@0&f" format specifier will round the output value to the third position after the decimal point, and remove the trailing '0' of the output result. This is the system default.

If it is positive, it specifies the number of decimal position to output the value with rounding, yet the trailing '0' will not be removed anyhow.

If it is negative, its absolute value specifies the number of decimal position to output the value with rounding, and the trailing '0' of the conversion result will be removed.

The "%0&f" format is used in the TGF as a format specifier for the value output conversion. It is also supported by the printf() function in the TCL runtime library, although it is not documented there. Note that the "%@0&f" format will always skip the leading space of the output conversion result.

The content of this variable also affects the screen coordinate display. If it is not zero, the screen coordinate display will be forced to use the "%@0&f" format for the value output conversion; otherwise, an internal fixed-point format (up to 4 decimal position) is used as usual.

@DO3DMODE

Attribute: Integer, On-off control, Default=OFF (0)

Description: The content of this variable controls the operation of certain commands in 3D/2D mode. This variable may be accessed by TCL expression only for the time being, since the 3D mode is not yet explicitly supported nor fully implemented.

Currently only **MOVE** and **COPY** commands recognize this variable and support the 3D mode operation. In 2D mode operation, the Z-offset is always zero, while in 3D mode, a Z-offset will be read and used in the transformation.

@DWGCOLOR

Attribute: Integer array of 16 colors, Volatile, Default=from INI file

Description: The content of this variable stores the 16 pen-color numbers used to override the @PENCOLOR setting whenever a DWG file is loaded or created as the current drawing file.

@DWGINPURGE

Attribute: Integer, Local, Default=0

Description: The content of this variable specifies whether to purge off unreferenced BLOCKs automatically from the DWG file being loaded during the DWGIN command processing, as described below:

- 0 Do not purge off unreferenced blocks, just load them in.
- 1 Query the operator for each unreferenced block before purging it.
- 2 Purge them all without any confirmation.

The DWGIN command will always remove the unreferenced anonymous blocks (such as erased dimension and hatching blocks) with any confirmation, since such blocks are gabages in the DWG drawing.

@DWGVERSION

Attribute: Integer, Local, Default=0

Description: The content of this variable controls the drawing version output by the DWGOUT command. The currently supported values are:

- 0** Specifies to output R10 compatible DWG files.
- 1** Specifies to output R11/R12 compatible DWG files.
- else** Reserved for future expansion, yet R11/R12 compatible DWG files will be output for the time being.

@DXFIOCNV

Attribute: Integer, On-off control, Volatile, Default=ON (1)

Description: The content of this variable controls the conversion of DIMENSION entities from the DXF or DWG file. If it is On, the DIMENSION entities from the DXF or DWG file (after R10) will be read in and converted into **TwinCAD's** Dimension entities. If it is Off, the DIMENSION entities will not be read, but the dimension image in the BLOCK will be read in instead. This variable affects **DXFIN** and **DWGIN** commands.

This variable's attribute was altered from Local to Volatile and its default value from OFF to ON, since Version 3 Release 3. Note that volatile variables are always initialized to default value at system start-up.

Updated since V3.1.054 The attribute of this variable is changed to local integer, since V3.1.054, and has an additional effect when its value is 2. If it is set to 2, all dimension entities from R12/R10 DWG file will always be converted by means of dimension picture analysis, and the original dimension attribute data stored previously by TwinCAD will be ignored.

@ECS_NX

Attribute: Double, Volatile, Readonly, Default=0.

Description: The content of this variable stores the X component of the normal vector of the current ECS plane setup (by **set_ecs()** function).

@ECS_NY

Attribute: Double, Volatile, Readonly, Default=0.

Description: The content of this variable stores the Y component of the normal vector of the current ECS plane setup (by **set_ecs()** function).

@ECS_NZ

Attribute: Double, Volatile, Readonly, Default=0.

Description: The content of this variable stores the Z component of the normal vector of the current ECS plane setup (by **set_ecs()** function).

@ECS_ORG

Attribute: Point, Volatile, Readonly, Default=<0.,0>

Description: The content of this variable stores the X-Y component of the origin of the ECS plane in WCS (specified by **ecs_org()** function).

@ECS_ORGZ

Attribute: Double, Volatile, Readonly, Default=0.

Description: The content of this variable stores the Z component of the origin of the ECS plane in WCS (specified by **ecs_org()** function).

@ECS_XDIR

Attribute: Point, Volatile, Readonly, Default=<1.,0.>

Description: The content of this variable stores the X-Y component of the X-axis direction vector on the ECS plane in WCS (specified by **ecs_org()** function).

@ECS_XDIRZ

Attribute: Double, Volatile, Readonly, Default=0.

Description: The content of this variable stores the Z component of the X-axis direction vector on the ECS plane with in WCS (specified by **ecs_org()** function).

@ECS_ZH

Attribute: Double, Volatile, Readonly, Default=0.

Description: The content of this variable stores the default entity elevation in the ECS plane for TEXT/DIMENSION entities (specified by **set_ecs()** function).

@EIDLEVEL

Attribute: Integer, Bit flag, Local, Default=0

Description: The content of this variable specifies the listing level of **ID** command, as described below:

- Bit 0** 1, Enable listing of Mother Entity
- Bit 1** 1, Enable listing of Referenced Block Content
- Bit 2** 1, Enable listing of Polyline segment in Block
- Else** Reserved, should be zero.

Enable Bit 1 and Bit 2 of **EIDLEVEL**, and pick up a block instance (INSERT) via **ID** command, is the only way to list out the content of a block definition without making an explosion of an INSERT of it.

@ELEVATION

Attribute: Double, Default=0.

Description: The content of this variable stores and specifies the default elevation for all subsequent new entities in creation.

@ELLIPSEARC

Attribute: Integer, On-Off control, Volatile, Default=ON

Description: The content of this variable specifies whether the system will use True Ellipse entity for an ellipse or elliptic arc in creation. If this variable is ON (which is recommended and default), the True Ellipse entity will be used; otherwise, the polyline approximation will be used. It is effective only for ELLIPSE and MVCOPY command operation.

Note: This variable's attribute was altered from Local to Volatile since Version 3, Release 3. All volatile variables are always initialized to their default values at system start-up.

@ENCRYPT

Attribute: Integer, On-off control, Local, Default=OFF (0)

Description: The content of this variable is used to enable or disable the encryption of the drawing work file in subsequent disk saving. If it is turned ON, then whenever the drawing is to save to disk in **WRK** file format, **TwinCAD** will ask for an encryption string, which will be used to encrypt the output data string to the disk file.

@ENTORDER

Attribute: Integer, bit-flag control, Local, Default=0

Description: The content of this variable is used to control the access order of entities. It is a bit flag control word and is described below:

Bit 0: Selection Set Option. If this bit is 1, a temporary order set of object will be created every time when the object SELECT operation is fulfilled. Objects accessed from this order set will follow the selection order from the SELECT operation. If this bit is 0, as a default, then no order set will be created; selected objects will be processed in the database order. This bit is effective to a selection operation only when it is set to 1 before the operation.

Else Reserved, and should be zero.

Note that setting bit 0 to 1 does not imply that all the command operations will use this resulted order set to process the selected objects. Since, to most of the commands, the processing order may be irrelevant to the purpose of operation, the use of this order set is thus command dependent. However, the SLIST returned by **getesel()** will definitely affected by the setting of this bit. The TCL application may take this advantage to obtain an order set of object selection.

@EXPERT

Attribute: Integer, Local, Default=1

Description: The content of this variable specifies the expert level of the operator. The default level is 1, which means the system will support the operations in the compatible way to all the previous version. The system level 0 is thus

reserved for less expert user than the one that the previous version assumed. Less expert level means more verbose message, more user intention confirmations and fool-proove checkings to the user operation. Higher expert level means short message and less confirmation on user intention.

@EXPLOLIMIT

Attribute: Integer, Local, Default=4096

Description: The content of this variable specifies the maximum limit number of new entities that can be created by the **EXPLODE** command in exploding an associatively hatched region.

@EXPLOTEXT

Attribute: Integer, On-off control, Local, Default=ON (1)

Description: The content of this variable controls the way how a text entity will be exploded by the **EXPLODE** command.

If it is ON, as by default, then the text entity will be totally exploded into elementary segments, which is compatible to the EXPLODE operation in the earlier versions. If it is OFF, then the text will not be fully exploded into segments but rather separated into Text entities containing each single character unit, while the image created by the text feature functions and the built-in symbols will still be exploded into segments.

@EXTDEXACT

Attribute: Integer, On-off control, Local, Default=OFF (0)

Description: The content of this variable controls whether to extend object exactly on the boundary objects in the **EXTEND** command. The default is OFF, which means that it is the imaginary geometries of boundary objects that are taken as the boundaries. If it is ON, then the extending operation will be compatible with the AutoCAD, and only when the object can be extended to a point on a selected boundary object, will it be extended. See also **EXTEND** command in Command Reference.

@EXTFNTYPE

Attribute: Integer, Read only, Volatile, Default=N/A

Description: The content of this variable stores the ID. code of the current loaded screen font extension overlay driver. Such drivers are overlay patches to the **TCAM Graphic Runtime** to handle the display font for non-English-like language, and are specified in the system initial file.

This variable is provided here for the TCL programmer to identify the extended screen font driver in their program, such that message texts can be properly chosen for the local language support. Currently supported ID. codes are listed below:

- 0x000** None (Null driver).
- 0x100** Traditional Chinese Font, BIG5 coded.
- 0x200** Simplified Chinese Font, GB2312 coded.
- 0x300** Korean Font, ET-Korean Coded.

0x400	Thai Font, TIS coded.
0x500	Japanese, JIS coded.
0x600	VietNameese, VNI coded.
Others	Reserved.

@FASTLTYPE

Attribute: Integer, On-off control, Local, Default=ON (1)

Description: The content of this variable is used to enable or disable the quick generation of screen line type. If it is turned OFF, the line type generation on the screen will be exactly as what is expected from the plotter output, which requires the system to take good care of all details of calculation, and which takes more time in the drawing regeneration. If it is turned ON, all line type generation to the screen will be done by the **TCAM Graphic Runtime** driver, which is very much faster than the exact one, however, in the expense of losing accuracy in details. It affects only the screen line type generation. The plotting output will not be affected by this variable setting.

@FILLETOPT

Attribute: Integer, Bit-flag, Default=1

Description: The content of this variable stores and controls the current setting of the fillet option. It is a bit-flag variable. The meaning of each bit is described below:

Bit 0	TRIM option, 1 to trim objects after filleting, 0 to generate the fillet arc without any trimming operation.
Bit 1	Break Circle option, 1 to break the circle into arc after filleting, and 0, otherwise.
Bit 2	Justify Fillet Segment option, 1 to justify the end point of the fillet arc on the segment of a polyline, and 0, otherwise.
Else	Reserved.

@FILLETRAD

Attribute: Double, Default=0.

Description: The content of this variable stores and specifies the current fillet radius used in **FILLET** command.

@FILLMODE

Attribute: Integer, On-off control, Default=ON (1)

Description: The content of this variable controls the solid color fill generation of drawing entities with solid color fill attributes. If it is ON, the solid color fill will be generated to the output devices (screen, plotter, and other possible devices) when output. If it is OFF, then the solid color fill attribute of drawing entities will be suppressed in output, i.e., no solid color fill will be generated.

@FONTOPTION

Attribute: Integer, bit flag, default=0

Description: The content of this variable specifies the options of font usage, as described below:

Bit 0.1: PFB font height calculation option

0, TwinCAD compatible, use original base line as the font base line and use X-Capital height as the font height.

1, AutoCAD compatible, use original base line as the font base line, but the font height is equal to X-Capital height overshoot minus the Base line overshoot.

2, Use original base line as the font base line, but the font height will be the X-Capital height overshoot.

3, Use the base line overshoot position as the font base line and use the X-Capital height overshoot as the font height.

Others, Reserved, and should be zero for future compatibility.

@GRIDCOLOR

Attribute: Integer, Local, Default=0

Description: The content of this variable specifies the palette color used for the grid point in the screen. If it is zero, the system will use an internal default color.

@GRIDUNIT

Attribute: Point, Default=N/A

Description: The content of this variable, being a point, specifies the grid units used in the X and Y axis, respectively. The grid units are used to control the spacing of grid dot when the GRID is turned ON.

@HIDEANONYM

Attribute: Integer, On-off control, Default=ON (1)

Description: The content of this variable control the hiding of anonym objects from the selection list of names. The default is ON, which means the names of such objects will be invisible to the operator in the name selection.

An anonymous object is usually an object media generated by the system to represent something, such as a group of hatching lines in a block. The naming of such an object is given by the system automatically, and may not be meaningful to the operator (though it may, once in a while). So, hiding it from the name selection list will make the operator feel easier.

All the anonymous objects generated by AutoCAD in DXF file format will also be recognized and controlled by this variable. This will be especially benefited, since all the dimensions read from the DXF files will be in anonymous blocks.

@HIGHLIGHT

Attribute: Integer, On-off control, Local, Default=ON

Description: The content of this variable specifies the system whether to highlight the objects in object selection or not. If this variable is ON, the objects in

selected state will be highlighted so that the operator may clearly identify them, during the object selection operation. However, if this variable is OFF, the objects being selected will not be highlighted. The default is ON. This variable is intended for TCL applications to fast up the command processing.

@INPSTATE Dos

Attribute: Integer, Bit-flag, Volatile, Readonly

Description: The content of this variable stores the current status of the Playback/Recording Mode. Each bit shows ON if it is 1, OFF if it is 0, as described below:

- Bit 0-1:** Keyboard input delay mode from KD0 to KD3.
- Bit 2-5:** Reserved.
- Bit 6:** Duplication Mode.
- Bit 7:** Verbose Mode.
- Bit 8:** End of Playback.
- Bit 9:** No-Delay Mode.
- Bit 10:** Recording Mode in Suspension (^] key controlled).
- Bit 11:** Playback Mode in Suspension (^[key controlled).
- Bit 12:** Stepping Mode.
- Bit 13:** Next Keyboard code pending due to stepping mode.
- Bit 14:** Sound ON (key click, beep on message display).
- Bit 15:** Message display OFF.

This variable is Readonly and Volatile, so it is not possible to setup an initial value for it via the use of prototype drawing. However, as it is sometimes desirable to have a different default setup other than 0, the system environment variable: "INPSTATE" is then used to setup its initial default. Use DOS command

```
drv>SET INPSTATE = nnn
```

to specify the default value, where *nnn* is an integer from 0 to 65535. Note that some of these control flags are not pre-settable, and will be masked off automatically.

@INSBASE

Attribute: Point, Default=(0,0)

Description: The content of this variable specifies the insertion base of the current drawing, also accessed by the **BASE** command.

@INSDMODE

Attribute: Integer, State control, Local, Default=0

Description: The content of this variable controls the display of a Block Instance (a reference of block). If it is 0, the block instance will be displayed in normal way (solid). If it is 1, the block instance will be drawn in dotted line, so that

you can recognize it from the display at once (valid only for those BYBLOCK entities in the block definition).

This is an obsoleted variable.

@LIMMAX

Attribute: Point, Default=N/A

Description: The content of this variable specifies the maximum drawing limit coordinate setting, used to restrict the area of grid display.

@LIMMIN

Attribute: Point, Default=N/A

Description: The content of this variable specifies the minimum drawing limit coordinate setting, used to restrict the area of grid display.

@LTMASK

Attribute: Integer, Bit-flag control, Volatile, Default=0

Description: The content of this variable specifies the entity selection mask by entity's linetype. Each bit of the integer value from bit 0 to bit 15, corresponds to a linetype ID from 0 to 15. If a bit is set to 1, the entity with the corresponding linetype number will be rejected in the object selection operation, provided that the selection masking is also enabled. The BYLAYER and BYBLOCK linetypes of an entity are always resolved to the actual linetype number used for it before doing the entity sieving by the mask. The linetype ID number after 16 are never masked off anyway.

@LTSCALE

Attribute: Double, Default=1.

Description: The content of this variable specifies an overall scale factor to apply to all linetype definitions specified in drawing unit. Note that this factor will not affect those linetypes specified in device unit (negative unit value).

Note that this is a global control variable. Changing this variable will change the pattern linetype generation in subsequent new plotting of drawing or drawing regeneration.

@MAXCHANGEP

Attribute: Integer, local, default=1

Description: The content of this variable specifies the maximum the maximum number of entities that are allowed to be changed by CHANGE command with the same change point at a time.

The CHANGE command, after V3R3, is modified to honor this variable and will issue confirmation prompt if the user has selected more than the setting number of entities and indicated a point to change them.

@MAXPVANG

Attribute: Double, Angle in unit of radians, Default=10°

Description: The content of this variable specifies the maximum interval of arc angle allowed for the plotter output to make segmentation over a circular curve segment. See **PLOT** command in Command Reference.

@MAXPVLEN

Attribute: Double, Default=5 mm

Description: The content of this variable specifies the maximum length of the line segment (in real world unit of MM) used in segmentation over a circular curve segment in plotter output. See **PLOT** command in Command Reference.

@MAXRECLINE

Attribute: Integer, Local, Default=0

Description: The content of this variable specifies the maximum line number allowed for the operation record buffer to keep the operation history records. If it is zero, the maximum line number will be set to 32767 internally.

The system will check whether the current number of lines in the operation history record buffer has exceeded this setting value at the top level command processing (and only at the top level command processing). Once the line number has exceeded this maximum setting value, the system will issue a warning message at the command area, and then continue to prompt the system command prompt for the next command. At this moment, nothing will happen.

However, at the end of the next command processing, the system will clear the buffer automatically if the situation still persists. This will give the operator a chance to issue the command to save the records if they are important, or to change the setting value to allow more lines if the records are to be preserved for a while, or to clear the record buffer explicitly so that the next command processing will be kept in the record buffer.

@MENUINUSE Dos

Attribute: Integer, ON/OFF state, Read only, Volatile, Default=N/A

Description: The content of this variable stores the status of the screen menu. If it is ON (1), the screen menu is enabled for use; otherwise, it is disabled and not ready for use. The control of the screen menu usage is determined by the system initial file (INI).

@MENUTYPE Dos

Attribute: Integer, State selection, Local, Default=0

Description: The content of this variable controls the behavior of the pulldown menu. If it is 0, which is the default, the pulldown menu will not automatically drop down until the operator pick at one of the menu titles. If it is 1, the pulldown menu will automatically drop down the menu whenever the cursor reaches at the menu titles. It is then called the fall-down menu or drop-down menu.

The operator may directly select the menu item from the fall-down menu without picking at the title bar first. The fall-down menu will be removed automatically if the operator moves the cursor outside of the menu.

@MINDONUT

Attribute: Double, Local, Default=0.01

Description: The content of this variable specifies the minimum diameter ratio (the inner diameter over the outer diameter) allowed for a 'DONUT' entity to be exactly converted from the DWG/DXF file. A 'DONUT' entity is a circle with a uniform line width (forming an inner diameter and an outer diameter) and expressed in the form of a closed polyline. It is usually created by AutoCAD's DONUT command.

The **DWGIN** and **DXFIN** command will try to recognize the 'DONUT' entity out off a closed polyline by means of analyzing its geometry. Once a 'DONUT' is recognized, it will be converted into a circle either with a wide-line property or a solid-fill state, depending on its diameter ratio. If the diameter ratio calculated is less than or equal to this setting value, it will be converted into a solid-filled circle; otherwise, it will be exactly converted into the equivalent circle with a uniform line width.

If this setting value is zero, then only when the inner diameter of a 'DONUT' is effectively zero will it be converted into a solid-filled circle. And, if this setting value is negative, then the 'DONUT' entity will always be converted into a circle with a wide-line property.

Note that the drawing generation speed of a solid-filled circle is much faster than a circle with a wide-line property. Also note that when the diameter ratio of a 'DONUT' entity is less than 0.01, it is effectively appeared as a solid-filled circle.

@MIRRTEXT

Attribute: Integer, On-off control, Default=OFF

Description: The content of this variable is used to enable or disable the mirroring of Text entities. If it is ON, a TEXT entity will be mirrored by the **MIRROR** command as if it is only a groups of lines and arcs. If it is OFF, the TEXT entity will be mirrored in such a way that its starting position and angle of direction are adjusted so that the text writing area are mirrored.

@MVSCALE

Attribute: Double, Default=0.

Description: The content of this variable control the overall scale of view mapping transformation. Default is zero, means no scaling will be done. It is effective in such commands like **ELLIPSE/Map**, which generates the ellipse by mapping a circle placed at a given entity plane and view from a specific direction, and **MVCOPY** that generate 3D-mapping image on selected entities.

Such transformation involves successive matrix multiplications. If the content of this variable is not zero, it will be taken as a constant scale factor to the matrix operations. The effect will be the objects being scaled in every dimensions.

For example, a line of 10 units in length will be transformed to line with a length of 8.16496... units for an Isometric Projection Mapping transformation. To obtain the same length after transformation (for ease of dimensioning), a scale factor of $\sqrt{3}/\sqrt{2}$ can be set to this variable before doing the Isometric Projection transformation using **MVCOPY**.

@NOEMPTYBLK

Attribute: Integer, On-off control, Local, Default=1 (ON)

Description: The content of this variable specifies the system whether to remove the empty block automatically during reading from or writing to a drawing file. The default is ON. An empty block is a drawing block containing nothing.

@OSMODE

Attribute: Integer, Bit flag control, Default=0

Description: The content of this variable controls the object snap running mode. It is reset and set by the **OSNAP** command. Useful bit flag informations are given below:

- Bit 0:** 1, ENDPoinT effective, 0 if not
- Bit 1:** 1, MIDPoinT effective, 0 if not
- Bit 2:** 1, CENTer effective, 0 if not
- Bit 3:** 1, NEAR effective, 0 if not
- Bit 4:** 1, QUADrant effective, 0 if not
- Bit 5:** 1, INTersection effective, 0 if not
- Bit 6:** 1, INSert/NODE point effective, 0 if not
- Bit 7:** 1, PERpendicular effective, 0 if not
- Bit 8:** 1, TANgent effective, 0 if not
- Bit 9-14:** Reserved, must be 0.
- Bit 15:** 1, NONe effective (return pick point), 0 if not

@PDMODE

Attribute: Integer, Bit flag and state selection, Default=0

Description: The content of this variable specifies the display type of the POINT entities. Useful bit flag informations are given below:

- Bit 0-4:** Select basic type of point marker:
 - 0 -- Single Dot
 - 1 -- Nothing
 - 2 -- Big cross of '+'
 - 3 -- Big cross of 'x'
 - 4 -- Vertical bar upward.
 - else -- Nothing
- Bit 5:** Add with a circle.
- Bit 6:** Add with a square box.
- Bit 7:** Solid color fill.
- Bit 8-15:** Reserved.

@PDSIZE

Attribute: Double, Default=N/A

Description: The content of this variable specifies the display size of the POINT entities. If the value is positive, it specifies the size in drawing unit. If the value is negative, it specifies the size in percentage value to the current window width. For example, a value of -5. specifies the size to be 5.0% of the current window width.

@PENCOLOR

Attribute: Integer array of 16 elements, Default=Depending on Initialization

Description: This is a special variable used to store the current color assignments of the 16 pen colors which are specified and initialized from the system INI file. It is used to save the color assignment of the pen with the drawing files, so that the display color will not change when the drawing files are ported to another working environment.

The usual reference to this variable in an expression can access its first element only. To access the whole array, you must use the *memmove()* to move its content to an integer array or to update its content by an integer array. Note that modifying this variable will affect the pen color display for subsequent drawing of entities.

@PICKBOX

Attribute: Integer, Local, Default=4

Description: The content of this variable specifies the size of the object snap box in units of pixel.

@PICKQUERY

Attribute: Integer, On-off control, Local, Default=OFF (0)

Description: The content of this variable controls the query function in single object picking. If it is turned ON, whenever there are multiple objects crossed by the target box, while only a single object is required, the system will enter the query mode for the operator to decide the one in desire. Valid only when selecting objects without any object snap directives. During the query mode, each of these possible objects will be height-lighted one by one, with a query message "[?]" in the command area. The operator may press <CR> or <Pick> key to confirm the choice. Any other key will skip it for the next one except the last one which makes no choice.

If it is turned OFF, as a default, the object snapping always return the first one crossed by the target box, in the drawing database order.

@PINPFILE Dos

Attribute: String, Filename, Volatile, Readonly, Default=NULL

Description: The content of this variable stores the current active INP file used in *Playback Mode*. The Playback mode may be setup at the command line when entering the system, or invoked by the **REPLAY** command.

@PLAYDELAY Dos

Attribute: Integer, Local, Default=0

Description: The content of this variable specifies the time delay modification count. This variable will be reset to zero whenever the playback mode is newly started (by **REPLAY** command). Its content will be incremented/decremented by one upon the keystroke of '-'/'+' during the playback mode. It is used as a bias to adjust the time delay for each subsequent data input read from the playback stream, units in tick (1/18.2 sec).

@PLOTOPTION

Attribute: Integer, Bit-flag control, Local, Default=1

Description: The content of this variable is used to control the plotting option, as described below:

- Bit 0:** End-point swapping optimization option, 1 to enable, and 0 to disable.
- Bit 1:** 1, to do the pen-width compensation in generating hatching lines over a solid fill area when the bit 7 is ON. Note that, this does not mean the solid fill area will be compensated with the pen-width.
- Bit 7:** Solid fill hatching option, 1 to direct hatch by scanning lines without any optimization and 0 to optimize the pen movement.
- Bit 13:** 1, disable the setup of the clipping region via Windows printing function call. 0, enable the clipping region setup. It is found that certain printer drivers may have problem in handling regional clipping, especially when prints to network printer with banding. In such a case, one can set this bit flag to turn off the driver's regional clipping. Note that TwinCAD will always clip the drawing before print.
- Bit 14:** 1, if the boundary frame of a solid fill area will be drawn, it will be drawn with thin line. 0, if the boundary frame will be drawn, it will be drawn with the width of the pen being specified. Effective only for PRPLOT command or commands to export image drawing.
- Bit 15:** 1, if the boundary frame of a solid fill area will not be drawn. 0, if it will be drawn with pen depending on the bit 14 flag. Effective only for PRPLOT command or commands to export image drawing.
- Else** Reserved, should be zero.

@POFSTCANG

Attribute: Double, Angle value in units of radians, Default=N/A

Description: The content of this variable controls the offset generation of a polyline. It specifies the minimum corner angle between two adjacent segments that are allowed to have a special rounding treatment. See **OFFSET** command in command reference manual.

@PPLSYMBOL

Attribute: String, Symbol name, Default=""

Description: The content of this variable specifies the name of the loaded symbol to represent the transparent layer of plot paper layout used in **PLOT** command for preview display purpose. If the symbol name is not specified, or the specified symbol is not loaded, the **PLOT** command will display the selected paper size in white rectangle only.

@PREVIEWOPT

Attribute: Integer, Bit-flag, Local, Default=0

Description: The content of this variable is used to specify the drawing preview options supported by the system. It is a bit-flag control variable. Useful bit flag values are described below:

Bit 0: Save Preview Slide Option. If this bit flag is ON, the system will save a preview slide of the current view to the disk whenever the drawing is saved (not by AutoSave, of course) to disk.

Bit 13: Preview in WBLOCK option. If this bit flag is ON, the Save Preview Image option is also effective to WBLOCK command and the preview image will contain the drawing extent view of the block.

Bit 14: Embed Preview Image Option. If this bit flag is ON, the system will embed the preview image (slide or what) into the drawing file, provided that the drawing file is in WRK file format. Otherwise, the preview image will be saved in a separate disk file with the same filename as the drawing file. This bit is effective only when a preview image saving option is enabled (currently, only Slide format).

Bit 15: Query Option. If the bit flag is ON, the system will query the user before saving the preview image. Otherwise, the preview image will be saved without notice. This bit is effective only when a preview image saving option is enabled.

Else Reserved, should be zero.

@QTEXTGENR

Attribute: Integer, On-off Control, Local, Default=OFF (0)

Description: The content of this variable control the option of quick spline text generation. If this option is ON, all the text segments that was built with spline arc (Quadratic B_ézier Curve) will be approximated each by 5 line segments, instead of the original two arcs, and that was built with Cubic B_ézier Curve, will be approximated by 8 line segments, instead of the original 4 arc segments.

The generation of such line approximation is much faster than the arc approximation, however, in the expense of less accuracy. Note that this option will not affect the result of **EXPLODE** and **PRPLOT**. Yet, the **PLOT** command is affected deliberately.

@QTEXTMODE

Attribute: Integer, On-off control, Default=OFF (0)

Description: The content of this variable controls the quick text display mode. If this mode is ON, then all the TEXT entities will be displayed as rectangular boxes at their text area.

The use of AUTOQTEXT function is preferred than the use of this function. See also AUTOQTEXT.

@RECSTATE

Attribute: Integer, On-off Control, Local, Default=ON (1)

Description: The content of this variable controls the **Operation History Recording Function**. If it is ON, all the operation history (command input and message output, etc.) will be saved in an internal file. The operator may type **Ctrl/Z** to view this history record.

@RINPFILE Dos

Attribute: String, Filename, Volatile, Readonly, Default=NULL

Description: The content of this variable stores the current active INP file used in *Recording Mode*. The Recording Mode may be setup at the command line when entering the system, or invoked by the **.RECORD** command.

@SARCTYPE

Attribute: Integer, Bit-flag value, Default=0

Description: The content of this variable controls the generation of the Spline Arcs (Dual-Arc Approximation of spline curve). The default value of this variable is 0, which is compatible to earlier version. The content of this variable is described below:

Bit 0-6: Bias Count if not zero, see explanation in later paragraphs.

Bit 7: 1 if bias backward, 0 if bias forward.

Bit 8: 1 if find Minimum Radius Difference (No biasing), 0 if controlled otherwise.

Bit 9: 1 if using Bisecting Angle Method, 0 if using Middle Point method (see later paragraphs).

Bit 10-13: Reserved, should be zero

Bit 14: 1 if generate complement arcs, 0 if standard.

Bit 15: Reserved, should be zero

If Minimum Radius Difference is enabled, then only Bit 14 is valid, and all else are ignored.

Affected commands are SARC, SPLINE, CSPLINE, BSPLINE, and the TCL function sarc(). Note that the ELLIPSE command is not affected. However, the method chosen to generate the SARcs for an ellipse approximation is changed from Middle Point to Bisecting Angle without any bias value. This

change will make the approximation more closer to the ellipse when the eccentricity is very large and the spline segment number used is small.

The Dual-arc Approximation technique is a good method to approximate a given plane curves. With two given points on the curve and their tangent directions also known, the approximation will produce two arc segments tangent to each other and passing the two points at the given directions. However, there are infinite number of solutions to meet such geometry requirement, if the closest approximation to the original curve is not taken into consideration.

Different method may be used with different property of curve to determine the optimal division point (where the two arcs joint) for the best approximation. An application that uses this technique must have known the property of the curve very well, and is responsible to develop an effective method to locate the optimal points. Such an example is the **INVGEAR** command that produces the precise shape of Involute Gears.

However, not knowing what the actual curve is to approximate, the SARC command or sarc() function must provide several pre-defined methods to determine the optimal point, so that the user may choose the one best for his application:

Minimum Radius Difference

This method is used to determine the division point such that the difference of the radii of the two arcs is a minimal for all possible solutions. The curve so approximated in the interval will be the 'flattest' approximation in that interval (but not necessarily the 'flattest' solution to the curve being approximated), provided that the division point is found in the interval. A division point is said within the interval, means that it lies inside the triangle formed by the control point and the two end points.

If this method is chosen, then it overrides all other options. No fine tuning is possible. Note that the division point may fall out off the interval (there can be no such point within that interval to produce a minimal of radius difference), and in such a case, it may result in wrap around of arcs or cusp.

This method is not recommended.

Middle Point

The basic division point is located on the line passing through the control point and the middle of the two end points. This division point can be biased (fine tuned) toward one of the selected end point.

Bisecting Angle

The basic division point is located on the line bisecting the angle from start point to the end point around the control point. This division point can be biased (fine tuned) toward one of the selected end points. If the division point is not biased, the line passing it to the center of the either arcs will be perpendicular to the line joining the two end points, and the ratio of the two radii (R_s/R_e) of the approximating arcs will be the nearest value to 1.

Fine Tune the Approximation

The approximation can be fine tuned with a bias count from 0 to 127 from the basic division point. The direction of the fine tuning can be specified either toward the start point or toward the end point. The more the value of the bias count, the closer the division point is moved toward the selected end point, and the smaller/larger (depending on which points) the radius of the arc will be resulted.

@SAVEBACKUP

Attribute: Integer, On-off Control, Local, Default=ON (1)

Description: The content of this variable controls the feature to rename an existing **WRK** file to **BRK** (Backup-wRK-file) file before updating it. Note that a file will be backup only once for its very first saving operation. All subsequent savings to the same file either by SAVE command or by Auto-save feature will not create the backup file again.

@SAVEFILE

Attribute: String, Filename, Local, Default=""

Description: The content of this variable specifies the output filename of the drawing file saved by the AutoSave Feature. If it is a null string, the system will save the drawing to the current drawing file. If it is not a null string, it will be used as the name of the output file. Note that there is no default extension for this filename. Also, if it contains a bad name, the AutoSave Feature will fail.

@SAVETIME

Attribute: Integer, Local, Default=0

Description: The content of this variable is used to control the AutoSave Feature with the elapse of time. It specifies the time interval in minutes since the first undoable command operation issued after the last drawing file saving operation that the system must do the saving of the drawing automatically.

Note that the saving operation occurs only in the top level command processing when a command is just finished or a new command is about to start. The system can not save the drawing if the system is in waiting for the user input, even when the elapse time is passing the limit.

@SELMASK

Attribute: Integer, Bit flag, Default=0

Description: The content of this variable specifies the selection mask used in the object selection operation (**SELECT**). It is effective only when the selection masking is enabled. Useful information of its bit fields are given as below:

- Bit 0:** 1, if a POLYLINE is masked off, 0 if it is acceptable.
- Bit 1:** 1, if a POINT is masked off, 0 if it is acceptable.
- Bit 2:** 1, if a LINE is masked off, 0 if it is acceptable.
- Bit 3:** 1, if an ARC is masked off, 0 if it is acceptable.
- Bit 4:** 1, if a CIRCLE is masked off, 0 if it is acceptable.
- Bit 5:** 1, if a 3DLINE is masked off, 0 if it is acceptable.
- Bit 6:** 1, if a TEXT is masked off, 0 if it is acceptable.
- Bit 7:** Reserved.
- Bit 8:** 1, if an INSERT is masked off, 0 if it is acceptable.
- Bit 9:** 1, if a DIMENSION is masked off, 0 if it is acceptable.
- Bit 10:** 1, if a REGION is masked off, 0 if it is acceptable.

- Bit 11:** 1, if a TAG is masked off, 0 if it is acceptable.
Bit 12: 1, if a SYMBOL is masked off, 0 if it is acceptable.
Bit 13: 1, if a 3DFACE is masked off, 0 if it is acceptable.
Bit 14-15: Reserved, must be 0.

@SHOWDIR

- Attribute:** Integer, On-off control, Volatile, Default=OFF (0)
Description: The content of this variable stores and controls the current status of SHOWDIR command (show entity direction mode).

@SNAPANGLE

- Attribute:** Double, Angle value in units of radians, Default=0.
Description: The content of this variable specifies the angle of rotation of the grid dot spacing in display. Note that setting this variable does not mean to change the current active snapping status. The variable **SNAPFLAG** must be accessed to turn the effect of this variable ON or OFF.

@SNAPBASE

- Attribute:** Point, Default=(0,0)
Description: The content of this variable specifies the origin of the grid dot spacing in display. Note that setting this variable does not mean to change the current active snapping status. The variable **SNAPFLAG** must be accessed to turn the effect of this variable ON or OFF.

@SNAPBETA

- Attribute:** Double, angle value in units of radians, Default=0.
Description: The content of this variable specifies the secondary snap axis direction angle relative to the first axis (as specified by "SNAPANGLE" relative to the world's X-axis). It is used to build a non-orthogonal coordinate system in the display. Effective only when its value is not zero, +/-90°, +/-180°, +/-270°.

When it is in effect, the following subjects will be affected:

1. *Grid in display*
2. *Cursor pointer position snapping*
3. *UCS coordinate system operation in coordinate inputs and display.*

Accompanied with the use of an UCS coordinate system, this additional feature will make the drawing of Isometric, Diametric, or Trimetric projection much more simple.

Note that when this variable is ineffective (as is by default), the UCS as well as the cursor snapping and grid display will assume an orthogonal coordinate system in operation.

The operator may access this variable from **DMODE** command.

@SNAPFLAG

Attribute: Integer, Bit flag, Default=0

Description: The content of this variable controls the system general status. Useful information of these bit fields are given below:

- Bit 0:** 1, if cursor snap mode ON, 0, if not, toggle by <Ctrl/B>.
- Bit 1:** Reserved.
- Bit 2:** 1, if ortho mode is ON, 0, if not, toggled by <Ctrl/O>.
- Bit 3:** 1, if ortho mode effect is enabled, 0, if not, individually controlled by internal command mode.
- Bit 4:** 1, if X-coordinate is fixed to snapbase, 0, if not, individually controlled by internal command mode.
- Bit 5:** 1, if Y-coordinate is fixed to snapbase, 0, if not, individually controlled by internal command mode.
- Bit 6:** 1, if SNAPBASE is effective in cursor snap mode, 0, if not, individually controlled by internal command mode (ON if SNAPBASE is not at origin). Note that if you set the SNAPBASE via the TCL expression, you must also set or reset this bit flag as well. It is only the DMODE command that sets/resets this bit flag automatically, based on the SNAPBASE coordinates.
- Bit 7:** 1, if rotation snap is effective in cursor snap mode, 0, if not, individually controlled by internal command mode (ON if SNAPANGLE is not zero). Note that if you set the SNAPANGLE via the TCL expression, you must also set or reset this bit flag as well. It is only the DMODE command that sets/resets this bit flag automatically, based on the SNAPANGLE value.
- Bit 8:** 1, if SNAPBETA is not zero and is effective, 0, otherwise.
- Bit 9-10:** Reserved.
- Bit 11:** 1, if current UCS setup is active, 0, if not, toggled by <Ctrl/U>.
- Bit 12:** 1, if object selection mask is enabled, 0, if not, individually controlled by internal command mode.
- Bit 13:** 1, if both object selection mask and layer mask are enabled in object selection operation, 0, if not, specified in SELMASK window operation.
- Bit 14:** Reserved.
- Bit 15:** 1, if grid display mode is enabled, 0, if not, toggle by <Ctrl/G>.

@SNAPGAMMA

Attribute: Double, angle value in units of radians, Default=0.

Description: The content of this variable specifies the third snap axis direction angle relative to the first axis (as specified by "SNAPANGLE" relative to the world's X-axis). It is used in the Axonometric Projection Control and is setup automatically by the AXOPLANE, ISOPLANE and PUCS command.

If this variable is zero, then the third axis is not setup.

@SNAPUNIT

Attribute: Point, Default=N/A

Description: The content of this variable specifies the spacing of the cursor snapping in cursor snap mode.

@SPLFRAME

Attribute: Integer, On-off control, Default=OFF (0)

Description: The content of this variable specifies whether to show the invisible frame of a 3DFACE.

@SPLINESEG

Attribute: Integer, Default=0

Description: The content of this variable specifies the number of additional data points on a parametric curve, such as a spline curve, to be added before applying the dual-arc-segment approximation. Affected commands are **BSPLINE**, **CSPLINE**, and **ELLIPSE** commands.

@SPLINETYPE

Attribute: Integer, State selection, Default=0

Description: The content of this variable specifies the type of B-spline to generate in the **BSPLINE** command. Supported values are given below:

- 0** Quadratic Bezier Curve combined with arc segments. The original arc segments will be preserved.
- 1** Cubic/Quadratic Bezier Curve combined with arc segments. The original arc segments will be preserved. The line segments between these arc segments are converted to Cubic/Quadratic Bezier Curves.
- 2** Non-rational Uniform Periodic B-Spline of order 3 (Quadratic)
- 3** Non-rational Uniform Periodic B-Spline of order 4 (Cubic)

The following describes the details of the curve generation by **BSPLINE** command when the **SPLINETYPE** is 2 or 3:

- ⇒ The polyline is used to specify the defining polygon vertices only. It does not matter with the segments (line or arc) it contains.
- ⇒ If the polyline is closed, the resulting curve will be closed with the same order of continuity.
- ⇒ If the number of vertex is less than 3, then no curve will be generated.
- ⇒ If the number of vertex is equal to 3, then a quadratic bezier curve will be resulted as a special case.
- ⇒ If the number of vertex is larger than 3, then Uniformed Periodic B-Spline curve (using uniform knot vector) will be calculated with the following boundary conditions:

1. If the polyline is open, the starting vertex and the ending vertex will be duplicated such that the starting point and the ending point of the B-Spline curve will coincide with the two given points.
2. If the polyline is close, the use of vertex will be cycled around the start point (ending point) such that the resulting B-Spline curve is closed and has the same order of continuity at the starting point.

B-Spline curve is a method to represent a space curve using defining polygon vertices. The **BSPLINE** command is thus used to realize the curve (approximation with arcs) upon its defining polygon vertices. It is used to 'generate' the curve, not to 'design' it.

The ACAD probably use the Open Uniform Basis functions to generate the B-Spline in its PEDIT/Spline command for open polyline. The shape of such spline is different from that generated by Periodic Basis Functions. The Open Uniform Basis Functions yield curves that behave most nearly like the Bezier Curve.

@SVARORDER

Attribute: Integer, Local, Default=0

Description: The content of this variable specifies the way how variables are appeared in the SYSVAR dialogue window, as described below:

- | | |
|-------------|---|
| 0 | To be appeared in the system internal default order (default) with limited system variables. |
| 1 | To be appeared in alphabetic descending sorted order with limited system variables. |
| 2 | To be appeared in the same appearing order as in the loaded INF file with limited system variables. |
| 3 | To be appeared in alphabetic descending sorted order with all the system variables. |
| Else | Reserved. |

Note: This variable affects only SYSVAR command. However, if its value is 2, the DIMVAR command will also be affected.

@SYMCOLOR

Attribute: Integer array, 16 elements, Default= All -1's

Description: The content of this variable specifies the color number assignment of local symbol colors. The local symbol colors define the colors assumed by the symbol elements (parts of a symbol) that have specification of local colors when they were built. A value of -1 means **BYLAYER** (by symbol).

The usual reference to this variable in an expression can access its first element only. To access the whole array, you must use the *memmove()* to move its content to an integer array or to update its content by an integer array.

@SYMESTYLE

Attribute: String, Style name, Default=""

Description: The content of this variable specifies the Extended Font style name for the text used in symbols. The initial default is nothing. This is a global setting of the text style used in symbols, but is a local setting for the text display in the Attribute Tags. So, be sure to set the desired one before inserting a symbol containing attribute tags.

@SYMSTYLE

Attribute: String, Style name, Default="STANDARD"

Description: The content of this variable specifies the English Font style name for the text used in symbols. The initial default is **STANDARD**. This is a global setting of the text style used in symbols, but is a local setting for the text display in the Attribute Tags. So, be sure to set the desired one before inserting a symbol containing attribute tags.

@SYSENDFLAG

Attribute: Integer, Readonly, Volatile, Default=0

Description: The content of this variable stores the current system termination flag. This variable is zero all the time till the system is about to terminate. It is used to inform the customized routine called at the system termination. The meaning of its value is given below:

- 0** System in Normal Execution State.
- 1** System is about to exit by END command, and the drawing is ALREADY SAVED.
- 2** System is about to exit by QUIT command.
- Else** Reserved.

The system will automatically invoke the command "**ENDING**" before exiting to the DOS, provided that the command "ENDING" is already defined and well loaded. The ENDING may check the system variable **SYSENDFLAG** to see whether the system is about to leave or not and why.

Note that if the system is terminated by **END** command, the drawing is saved before the ENDING is called. This is because the saving of the drawing is done at the command level where the **END** command is executed, while it is at the final point (Very Top level) that the ENDING is called. So, if the ENDING command should have modified the drawing, it may save it again by itself (using *command("SAVE...")*, of course).

@SYSOPTION Win

Attribute: Integer, Local, Default=0.

Description: The content of this variable stores the local option of the system operation, as described below:

- Bit 0** 1, if the ALT-key will be interpreted according to the current menu file definition; 0, if the ALT-key will be interpreted by Windows for the short-cut key activation.
- Bit 1** 1, if the use of the scroll bar in drawing view window is disabled; 0, if it is enabled (default).
- Else** Reserved, and should be zero for future compatibility.

If you enable the use of the ALT-keys from the menu definition, you will not be able to access the pull down menu by direct ALT-key; however, you may press a single ALT key and then release it, then press the specific alphanumeric key to access the pull down menu.

If the use of the scroll bar in drawing view window is being disabled, the drawing regeneration will generate the view required only by the current size of the view window. This would be faster, since all those drawing vector fall off the view window will be clipped off from the display list.

However, if the use of the scroll bar is being enabled, the drawing regeneration will assume that the width and height of the display surface is 4096 by 4096 pixels, and the user may scroll the view over each part of the display surface. In such a case, more drawing vector are generated and is thus much slower.

@SYSPROFILE Win

Attribute: String, Readonly, Volatile

Description: The content of this variable stores the filename of the current active system profile in use.

@SYSVERSION

Attribute: Integer, Readonly, Volatile

Description: The content of this variable stores the version number of the running TwinCAD.

@TCAMDXFEXT

Attribute: Integer, On-off control, Local, Default=OFF (0)

Description: The content of this variable controls whether the DXFOUT command will produce DXF file with TCAM/DXF-Extension or not. The default is OFF.

If this variable is OFF, the **DXFOUT** command will produce DXF file conforming with the AutoCAD specification and should be read in AutoCAD without any obvious offending problem. However, if the drawing containing informations more than the DXF can hold directly, these informations may be lost. For example, Multiple Block Instance in Polar Array will be output in each separate copy of block instance; the polar array informations are lost. Another examples are the character distance and circular text radius, these can not be output in standard DXF file; the text in DXF will appear without character distance adjustment and in linear writing only.

If this variable is ON, the **DXFOUT** command will produce DXF file with the TCAM/DXF-Extension, which extends the DXF format to hold additional geometry informations directly. The DXF file such produced may not be properly read in by AutoCAD and by other S/W that read DXF file with strict checking rules. However, the drawing informations will not lose due to the deficiency of the original DXF specification. These additional informations included in such extensions may be important to certain applications that read drawings in DXF format. All newly updated TCAM products will support the DXF with this extension. See **TwinCAD** supplement documentation on the specification of the TCAM/DXF-Extension.

@TXTLTYPE

Attribute: Integer, On-off control, Default=ON

Description: The content of this variable specifies whether to allow linetype pattern generation on all TEXT entities or not. If this variable is set to ON, the strokes of the TEXT entities with a linetype other than CONTINUOUS will be generated using the specific linetype accordingly. If it is OFF, then the text strokes will always be CONTINUOUS. For ACAD-compatibility, set this variable to OFF.

As ACAD does not have the capability to generate text fonts using specified patterned lines, while **TwinCAD** has provided this capability as a basic property of a TEXT entity, the drawing imported from a DWG file may look different if there are TEXT entities specified with a linetype other than CONTINUOUS.

@TGFTYPE

Attribute: Integer, Volatile, Readonly, Default=N/A

Description: The content of this variable stores the Language Identification number of the current load Text Generation File, as below:

0	English Text
1	Traditional (Standard) Chinese Text.
2	Simplified Chinese Text.
3	Korean Text.
4	Thai Text.
5	Japanese Text.
6	Vietnamese Text.
Else	Reserved.

@TG_FILE

Attribute: String, Filename, Volatile, Readonly, Default=NULL

Description: The content of this variable stores the name of the current secondary *Text Generation File* setup by the **tg_open()** functions.

@THICKNESS

Attribute: Double, Default=0.

Description: The content of this variable specifies the default thickness value for subsequent entities of new creation.

@TXTBOXGAP

Attribute: Double, Default=-0.25

Description: The content of this variable specifies the default gap distance between a text and its virtual bounding box, used in **HATCH** command. If the value specified in this variable is positive, then it specifies the gap distance in

drawing unit directly. However, if the value is negative, its absolute value then specifies the ratio of the gap distance with respect to the text height. If the value is zero, it means no gap at all, and the bounding box will start from the base line to the text height.

The system default value is **-0.25**, which means the gap distance will be 1/4 of the text height.

Note that the specification of this variable will not affect the display result of **QTEXT** mode. The **QTEXT** mode will display the text bounding box without any gap distance calculation (so as to be fast).

@TXTCSPACE

Attribute: Double, Default=0.

Description: The content of this variable specifies the default character space adjustment for subsequent creation of TEXT entities. A positive value specifies absolute spacing (in addition to the original one defined by the font) in drawing unit, a negative value specifies a relative spacing as a ratio over the text height, and a zero value will disable the space adjustment function.

@TXTESTYLE

Attribute: String, Style name, Default=""

Description: The content of this variable specifies the style name of the Extended Font used for subsequent creation of TEXT entities.

@TXTHEIGHT

Attribute: Double, Default=N/A

Description: The content of this variable specifies the default text height used for subsequent creation of TEXT entities.

@TXTOBANGLE

Attribute: Double, Angle value in units of radians, Default=0

Description: The content of this variable specifies the default text oblique angle used for subsequent creation of TEXT entities. The valid range of the oblique angle is from -360° to $+360^\circ$. But, the effective oblique angle value must be within the range from -90° to $+90^\circ$. The value outside of this effective value range is used to carry additional controls:

1. The sign of the angle value determines the oblique direction.
2. If it is in vertical writing, the angle should be increased by 90° in the same direction.
3. If it is to oblique from the baseline, the angle should be complemented by the full 360° .

@TXTOBX

Attribute: Integer, On-off control, Default=OFF (0)

Description: The content of this variable specifies the default oblique text control state for subsequent creation of TEXT entities. If it is ON, the oblique angle will be

taken as an angle to oblique the font from its baseline, instead of its vertical axis. It is effective only when the current default text oblique angle **@TXTOBANGLE** is within the range from -180° to $+180^{\circ}$.

@TXTROTCCW

Attribute: Integer, On-off control, Default=OFF (0)

Description: The content of this variable specifies the default vertical text control state for the subsequent creation of TEXT entities. If it is ON, the subsequent new TEXTs will be created in vertical writing style; that is, each text character font will be rotated by 90° CCW. This variable is effective only when the current default text oblique angle **@TXTOBANGLE** is within the range from -90° to $+90^{\circ}$.

@TXTROWDIST

Attribute: Double, Default=0.

Description: The content of this variable controls the default text row distance by the following rules:

1. If it is zero, which is the default, the default text row distance will be taken from the text font via the linefeed code. This provides the compatible mode with the earlier version. However, should the definition of the linefeed code in the text font file be missing, a text row distance of 1.5 times of the text height will be assumed.

2. If it is greater than zero, the value will be directly taken as the absolute row distance for subsequent text generation.

3. If it is negative (less than zero), its absolute value will be taken to specify the ratio of the text row distance over the text height. That is, the text row distance will be the text height multiplied by the absolute value of this variable.

Note that the default text row distance will be scaled down and up automatically by the **Shift-in/out** functions ('{' and '}').

It is recommended to set this value to a negative value, say -1.5 or -2.0, since some of the text font files may not have a proper definition for the linefeed codes.

@TXTSTYLE

Attribute: String, Style name, Default=""

Description: The content of this variable specifies the style name of the English Font used for subsequent creation of TEXT entities.

@TXTSUBFAC

Attribute: Double, Default=1.

Description: The content of this variable specifies the scale factor of the text height for the text superscript and the text subscript. This is a global control variable. Changing this variable will cause all TEXT entities containing superscripts or subscripts to change their size in display, after regeneration.

@TXTWIDTHF

Attribute: Double, Default=N/A

Description: The content of this variable specifies the default text width factor for the subsequent creation of TEXT entities.

@UCSCONTROL

Attribute: Integer, Bit-flag, Default=0

Description: The content of this variable controls the display of UCS-icon and other related operation. This variable is an integer bit flag variable, with a default value of zero. Currently defined bit flags are described below:

Bit 0 1, Disable rotation and oblique feature on UCS-Icon, valid only when the UCS-Icon is an external symbol. If the UCS-Icon is the system generated vectors, it will always be rotated and oblique with the UCS-Axes directions.

Bit 0 1, Always display the UCS-Icon at lower left corner, provided that the size of the UCS-Icon is given relative to the screen window. If an absolute size value is given for the UCS-Icon, it will always be displayed at the UCS-origin in the drawing's modal space.

Bit 2-15 Reserved, should be zero.

@UCSORG

Attribute: Point, Default=(0,0)

Description: The content of this variable specifies the current **UCS** (User Coordinate System) X and Y origin in **WCS** (World Coordinate System). Note that setting this variable changes only the UCS origin. It does not alter the current UCS active status. To turn ON or OFF the UCS, access the **SNAPFLAG** variable.

@UCSORGZ

Attribute: Double, Default=0

Description: The content of this variable specifies the current **UCS** (User Coordinate System) Z origin in **WCS** (World Coordinate System).

@UCSSIZE

Attribute: Double, Default=N/A

Description: The content of this variable specifies the display size of the **UCS** icon. If the **UCS** icon is a symbol reference, this variable specifies the scale factor for it if the value is positive, or a relative scale factor (percentage value) to the window width, assuming the symbol was prepared in a unit square of one drawing unit.

However, if the **UCS** icon is not specified as a symbol reference, since it will be displayed as a POINT entity, the content of this variable functions as the **PDSIZE** to a POINT entity. See also **PDSIZE**.

@UCSSYMBOL

Attribute: String, Symbol name, Local, Default=""

Description: The content of this variable specifies the name of the loaded symbol to represent the **UCS** icon in display. If the symbol name is not specified, or the specified symbol is not loaded, the **UCS** icon will be displayed as a POINT entity with a display type value of **0x22** (a circle with a '+' at center).

@UNDOCTRL

Attribute: Integer, Local, Readonly, Default=0

Description: The content of this variable specifies the current option of undo control. This variable is modified only by UNDO command. The initial default is always 0. The meaning of each bit-flag is given below:

Bit 0	1, Undo feature Disabled. 0, Enabled.
Bit 1	1, in Single Undo Mode. 0, No Limits.
Bit 2	1, Include view window change commands. 0, Do not include view window change commands.
Else	Reserved, should be 0

@UNITALPHA

Attribute: Double, Default=0.

Description: The content of this variable specifies the length of the unit vector used for the first axis of the PUCS/UCS coordinate system in modal space drawing unit. If this value is less than or equal to zero, then the unit is ONE, which means in the same unit as that in modal space.

This variable also specifies the foreshortening factor for the first axis after a Parallel Projection transformation.

@UNITBETA

Attribute: Double, Default=0.

Description: The content of this variable specifies the length of the unit vector used for the second axis of the PUCS/UCS coordinate system in modal space drawing unit. If this value is less than or equal to zero, then the unit is ONE, which means in the same unit as that in modal space.

This variable also specifies the foreshortening factor for the second axis after a Parallel Projection transformation.

@UNITGAMMA

Attribute: Double, Default=0.

Description: The content of this variable specifies the length of the unit vector used for the third axis of the PUCS/UCS coordinate system in modal space drawing unit. If this value is less than or equal to zero, then the unit is ONE, which means in the same unit as that in modal space.

This variable also specifies the foreshortening factor for the third axis after a Parallel Projection transformation.

@USEAUXMENU Dos

Attribute: Integer, State selection, Volatile, Default=N/A

Description: The content of this variable specifies the usage of the auxiliary sub-command menu. It is initialized from the system initial file. Setting the content of this variable will change the property of the sub-command menu. The meaning of each supported values are given below:

- | | |
|------|--|
| 0 | Disable the use of sub-command menu. |
| 1 | Enable the use of sub-command menu, but co-exist the sub-command menu with the screen menu. It will be active in the screen menu area (push the screen menu down), provided that the command in execution is not activated from the screen menu. Note that if the screen menu is disabled, it will also be disabled. |
| 2 | Enable the use of sub-command menu, but co-exist the menu with the function key display. When it is active, the function keys will be redefined by the sub-command option. |
| 3 | Enable the use of sub-command menu in its own specific window, as specified in the system initial file. |
| 4 | Same as 3, but the sub-command menu will not be displayed in double width anyway. |
| else | Disable the use of sub-command menu anyway. |

@USERVARINF

Attribute: String, Volatile, Default=""

Description: The content of this variable specifies the name of the information file for user variables. A user variable may be created by TCL function.

@USE_EMODE

Attribute: Integer, On-off control, Default=OFF (0)

Description: The content of this variable specifies whether to use the property from the Entity Mode Setup (EMODE command) for entities created by certain editing commands. The default is OFF. The setting of this variable affects only those editing commands that will be or has been enhanced to recognize it and so as to have a useful application with the said effect.

For example, the OFFSET command will take the property from the referenced entity for the newly created entity if the USE_EMODE is OFF, and use the default property from EMODE setting for the newly created one if the USE_EMODE is ON. The default properties include layer, color, linetype, elevation and extrusion thickness.

Another example is the COPY command. When the USE_EMODE is ON, the COPY command is able to copy entities from one layer to another layer with new color, linetype, elevation and extrusion thickness.

@VWNDMAX

Attribute: Point, Read only, Default=N/A

Description: The content of this variable stores the maximum coordinates of the current view window. It is updated by the **ZOOM** and **PAN** command.

@VWNDMIN

Attribute: Point, Read only, Default=N/A

Description: The content of this variable stores the minimum coordinates of the current view window. It is updated by the **ZOOM** and **PAN** command.

@VZCENTER

Attribute: Double, Default=0

Description: The content of this variable specifies the Z-center coordinate of the 3D view window box. This setting affects the view point generated by the **VPOINT** command.

@VZSCALE

Attribute: Double, Default=1.

Description: The content of this variable specifies an additional scale factor on the Z-axis coordinate when in 3D viewpoint. It affects the view in 3D view point generated by **VPOINT** command.

@WIDAUXMENU Dos

Attribute: Integer, window ID., Read only, Volatile, Default=N/A

Description: The content of this variable stores the graphic text window ID. number of the Auxiliary Sub-command Window, if it is opened during the system initialization.

@WIDCMDAREA Dos

Attribute: Integer, Window ID., Read only, Volatile, Default=N/A

Description: The content of this variable stores the graphic text window ID. number of the Command Text Window which is always opened during the system initialization for command prompt.

@WIDRAWING Dos

Attribute: Integer, window ID., Read only, Volatile, Default=N/A

Description: The content of this variable stores the graphic text window ID. number of the Main Drawing Window, which is always opened during the system initialization for the CAD drawing graphic.

@WIDSCRMENU Dos

Attribute: Integer, window ID., Read only, Volatile, Default=N/A

Description: The content of this variable stores the graphic text window ID. number of the Screen Menu Window, if it is opened during the system initialization.

@WIDSTATUS Dos

Attribute: Integer, window ID., Read only, Volatile, Default=N/A

Description: The content of this variable stores the graphic text window ID. number of the Status Line Window, which is always opened during the system initialization for the system status display.

@WORKEMAX

Attribute: Point, Read only, Default=N/A

Description: The content of this variable stores the maximum coordinates of the current work drawing. It is updated by **ZOOM/Extent** and also by new entities in creation. Note that the value stores here needs not to be correctly reflecting the actual drawing extent, since the editing operation may cause the actual drawing extent change.

@WORKEMIN

Attribute: Point, Read only, Default=N/A

Description: The content of this variable stores the minimum coordinates of the current work drawing. It is updated by **ZOOM/Extent** and also by new entities in creation. Note that the value stores here needs not to be correctly reflecting the actual drawing extent, since the editing operation may cause the actual drawing extent change.

@WORKSTATUS

Attribute: Integer, Read only, Volatile

Description: The content of this variable stores the status of the current work drawing file, as described below:

Bit 0: 1 if the work drawing has been modified since the last explicit saving, or new/loading or new/creation.

Bit 1: 1 if the work drawing is readonly.

Bit 2: 1 if the work drawing is being locked by TwinCAD.

@ZESCALE

Attribute: Double, Local, Default=1

Description: The content of this variable specifies the additional zoom factor to the drawing extent when the **ZE** command is executed. The default is 1.0, which means the view will be zoomed exactly to the drawing extent. Valid range of this value is from 0.5 to 1.0, and a value not less than 0.9 would be more desirable.

@ZTESTEPS

Attribute: Double, Local, Default=0.000001

Description: The content of this variable specifies the epsilon value for internal floating point value equality test. It should be within the range from 0.00001 to 0.00000000000001. Don't change this value, unless you have a good reason.

PART 3

TCL Runtime Library V3.10

Overview of TCL Runtime Library V3.10

This part of the document describes the supported functions in the **TCL Runtime Library** built in the **TwinCAD**. All the functions in this runtime library can be directly invoked from the command line as a part of an expression.

As the **TCL** is a case sensitive language (as is the C/C++ language), all the character cases of the name of these runtime functions must be exactly the same as those shown in this document, however, with one exception. Those functions that are described as **Intrinsic Functions** are case insensitive to the system. Both upper case and lower case characters of the name are accepted by the system.

The **Intrinsic Functions** are functions that are mostly referenced in expressions of arithmetics for mathematic calculation. Since the **TCL** expressions can be directly entered from the command line, the design of such functions must take the operator's convenience into considerations.

For example, the operator usually uses the degree system to specify an angle, so instinctively typing the expression: $10*\sin(30)$ would certainly be preferred than typing the expression: $10*\sin(dtr(30))$, if the **sin()** function accepts argument in units of degree instead of radian. And also, once in a while, the operator may have pressed the *Caps Lock* key on the keyboard, and thus the foregoing expression will totally be in upper cases. It would be a great surprise if the operator should receive an error message for that.

While the possibility of porting a TCL program to a C/C++ program in the future is also considered, accepting the upper case characters for those Intrinsic Functions provides a feasibility in solving the problem. A third party program written in TCL may follow strict rules in using the TCL library functions, so that the porting problem can be reduced to a minimum extent. For example, since the **sin()** from the standard C library accepts argument in radians, a strict TCL program should use the upper case function **SIN()** instead of the lower case

sin(), such that, in the future, the porting needs only a macro definition of ***SIN()*** to define a sine function that accepts an argument in units of degree.

The design of Intrinsic Functions does not cover only the arithmetic and mathematic functions, but also some other type of functions. Among which, there are 8 notable ones that can be classified as ***Type Casting Functions***. These are ***A()***, ***C()***, ***E()***, ***I()***, ***P()***, ***L()***, ***S()*** and ***V()***, in corresponding to the TCL registers of ARC, CIRCLE, ENTITY, integer, POINT, LINE, STRING and double data type, respectively (so these may also be called the ***Register Functions***). As specified in the manual, the name of these 8 functions are reserved as predefined identifiers, and may not be overloaded by user defined functions.

Type Casting, in other words, is an explicit request of type conversion. The arguments being casted to the specific type must be converted the specified type of data in return. Unlike the usual C/C++ type casting operator, the TCL's Type Casting Functions accept more than one arguments. The details of the type conversion process thus depends on the passed arguments.

For example, the expression: *V(P1,L1)* will return the distance from the point P1 to the line L1, and the expression: *P(P1,L1)* will return the base point on the line L1 with respect to the point P1.

The Type Casting Functions are also invoked automatically by the pure assignment operator (=) to convert the right side expression (single expression or comma expressions) to the data type same as the L-value. So, the expression: *V1=V(P1,L1)*; can be written as *V1=P1,L1*; without explicitly using the Type Casting Function. Now, you can see the power of such Type Casting Functions and the convenience they have brought to the operators in entering expressions at the command line for a quick solution.

In the following paragraphs, all the supported functions are described in alphabet order. The details of type conversion upon different arguments of these Type Casting Functions are also discussed in this document.

An Important Notice to TCL programming:

The *TCL Kernel Intrinsic Function* (such as ***sin()***, ***cos()***,...) can not be overloaded (redefined) by local defined function in current version!

This can be taken as a BUG, as the specification of TCL does not assume the existence of any pre-defined functions of which the names are reserved from the local use. However, since the problem is seldom arisen from the programming activities and can be avoided easily, and while the parsing of an expression will be faster without the overload checking on these intrinsic functions, this bug is not fixed in current version.

Compatibility between Windows and Dos Version

TwinCAD is a transplanted version from its DOS counterpart. All the TCL library functions from the DOS version are also transplanted, so that a TCL application works for the DOS version will also work for TwinCAD in Windows environment. However, since the two platforms are so different, some functions that relate with graphic user interface may have different features and support. These will be noted in this document where appropriate.

New functions and features are also added to enrich the TCL programming in Windows environment. These are not backward compatible with the DOS version, and will also be noted.

Unless otherwise specified, any function should work as documented for both DOS version and Windows version.

a or A

Intrinsic function to return an arc by geometry definition.

ARC **A** (*argument...*)

Var argument Variable number of arguments of predefined type. Different number of arguments of different type will specify different definition of an arc.

Return Arc of the definition results.

There are many ways to define an arc. To provide each of them with separate functions is cumbersome in these functions's naming as well as in referencing them. Recalling that the operator may directly enter a valid TCL expression to the command line and utilize the power of TCL expression to facilitate his drafting operation, lengthy naming would certainly be unacceptable. This is one of the major reason to provide a general arc geometry definition function as **A()** and take it as a Type Casting Function.

The methods used to define an arc depends on the arguments passed to the function, and are described below:

- ent** **Define an arc by directly loading it from a given entity handle** that points to an arc entity in the drawing database, or from another ARC entity.
- pc,ps,pe** **Define an arc by giving the center point, the start point, and the ending point of the arc.** The radius of the arc will be the distance from the start point to the center point. The spanning angle and direction are taken from the starting point **counter-clockwise** around the center point to the ending point. The ending point needs not to be exactly on the arc, since it is the direction angle taken. All the three arguments must be of POINT type. The first argument may be an entity handle that points to a POINT entity.
- ccl,ps,pe** **Define an arc on a given circle or arc** (the first argument), **with the start point and ending point specification** (the second and the third arguments) to specify the starting angle and ending angle of the arc. The spanning angle and direction are taken from the starting point **counter-clockwise** around the center point to the ending point. Both the two given points need not to be exactly on the arc or circle, since they are used to calculate the angle direction only. The first argument may be an entity handle that points to a circle or an arc.
- pc,ps,span** **Define an arc by giving the center point, the start point, and the spanning angle of the arc.** The radius of the arc will be the distance from the start point to the center point. The first argument may be of POINT type or an entity handle that points to a POINT entity. The second argument must be of POINT type. The third is a double value given in units of degrees (Tagged with an 'A' is also accepted).
- ccl,sang,eang** **Define an arc on a given circle or arc** (the first argument), **with a given start angle and a given ending angle in counter-clockwise direction** (the second and the third arguments). The first argument may be of CIRCLE or ARC type or an entity handle that points to a circle or an arc.

To find the arc segment passing through three points, include the following macros to your programs:

```
#define a3pccw(ps,pm,pe) A(c(ps,pm,pe),ps,pe)
#define a3p(ps,pm,pe) \
((ent_area(ps,pm,pe)>0)?a3pccw(ps,pm,pe):a3pccw(pe,pm,ps))
```

The macro **a3pccw()** returns the arc segment on the circle passing through the given three points, starting from the first point (*ps*) to the third point (*pe*) counter-clockwise. And, the macro **a3p()**, using the **a3pccw()**, implements the function that returns the arc segment starting from the first point (*ps*), through the second point (*pe*), and ending at the third point (*pe*). Note that the '\n' in the **a3p()** macro is used to indicate the continuation of the line at the next line. This function is not supported in TCL, it is used here mainly because the printed line can't hold the definition as a whole in one line. User must enter the definition as one line without the '\n' mark to make it work in TCL.

Note that this intrinsic function will be invoked automatically by the pure assignment operator (=) when the data type of the L-value is ARC. For example, the expression

```
A1 = P1,P2,P3
```

will be equivalent to the expression

```
A1 = A(P1,P2,P3)
```

where the **A()** function is called by the assignment operator to evaluate the comma expressions to the right of it.

abs

Intrinsic function to take absolute value of the argument.

Var abs (val)

Var val Integer, long or double value of which the absolute value will be returned. **POINT** data is also accepted as a complex number to calculate its absolute value.

Return Absolute value of the argument. The returned data type is the same as the argument. However, if the argument is of POINT type, the returned value will be of double data type.

One way to calculate the distance between two points is to apply this **abs()** function over their difference such as *abs(p2-p1)*. Also, to obtain the direction unit vector from p1 to p2, use the expression: *(p2-p1)/abs(p2-p1)*.

acos

Intrinsic function to calculate the trigonometric arc cosine function of the argument value in units of degree.

double acos (val)

double val Cosine value lies within -1. and 1.

Return The angle value in units of degree, which lies between 0° and 180°.

Note that the result of this function is in units of degrees. It is the **acosr()** that returns the result in radians. This is different from the one in the standard C library. If the argument

is not in the valid range from -1. to 1., it will cause a run-time error: *Operand Value Over Range*.

acosr

Intrinsic function to calculate the trigonometric arc cosine function of the argument value in units of radian.

double acosr (val)

double val Cosine value lies within -1. and 1.

Return The angle value in units of radian, which lies between 0 and pi. See also **acos()**.

add

Intrinsic function to implement the '+' operation.

Var add (args...)

Var args Variable number of arguments of applicable data type. The minimum number of arguments are 2, while the maximum number are 15. Type conversions are performed automatically from left to right in sequence.

Return Result of the summation, depending on the type of arguments.

This is the intrinsic function to implement the '+' operation in the expression. In other words, the expression: a+b+c+... can be written as **ADD(a,b,c,...)**. Note that both upper case and low case of the function names are accepted. The type of operands can be integers, doubles, POINTs, and character strings. Type conversions from integer value to double are performed automatically when they are mixed in the operation. See the table below:

Operand1	-	integer	double	string	POINT
Operand2		-----			
Integer		integer	double	error	error
Double		double	double	error	error
String		error	error	string	error
POINT		error	error	error	POINT

Note that the result of string summations will be the concatenation of these string operands. The **strcat(s1,s2)** from standard C library can be defined as *(s1=add(s1,s2))*, or *(s1+=s2)* in **TCL** using macro.

For portability consideration, use such function calls to add strings or POINTs.

add_sym

Add or update a specific symbol to a specific symbol library.

int add_sym **(smbfile,symname,insbase,entlist,taglist,symext)**

- char *smbfile*[]** Name of the symbol library file.
- char *symname*[]** Name of the symbol to update or to create. It will always be changed to upper cases.
- POINT *insbase*** Insertion base point of the symbol.
- SLIST *entlist*** Selection list of the entities that define the symbol image.
- SLIST *taglist*** Optional selection list of Text entities that define the tag attributes to add to the symbol definition. If this argument is not given or of integer type, the function
- LINE *symext*** Optional LINE data specifying the bounding box of the symbol. If this is not given, the geometric extent of the selection list will be used as the bounding box of the symbol.
- Return** -1, if an open error is encountered on the symbol library file, 0, if no valid symbol being created, and else the number of drawing vectors written out for the newly created symbol.

This function will search through the directory path specified in the TCADPATH environment variable for the specific symbol library, if it can not be found from the path it is specified. However, if it can not be found, this function will not create it but return -1. To create a new symbol library, use **symlib()** function.

The symbol image carried by the entities in the selection list given by **entlist** will be converted to symbol vectors under the same rules followed by the **SYMLIB** command. An optional entity list of TEXT entities can be given in the argument **taglist** to specify the variable texts using tag attributes. The content of the text string to specify a tag attribute must be given in the format:

{??}tag-name{:initial text string}

where the optional characters "??" are checked out only for compatible reason. Such text entities will be converted into equivalent TAG entities and added into the symbol definition.

The basic size of a symbol is defined by the drawing extent found from the defining entities. The application may use the POINT entities as the pivot point to set the drawing extent, since the POINT entities will not contribute to the symbol image. Or, you may directly supply an optional LINE data to specify the basic size of the symbol (as the minimum extent and the maximum extent). If this argument is given, it will override the default.

The symbol such created will be directly written into the symbol library file without affecting anything from the drawing database. This is different from the **SYMLIB** command. However, if there is already a symbol of same name loaded in the system, **add_sym()** will update it as well, no matter where the symbol was loaded from. The user may regenerate the drawing to observe the change of symbol image in the drawing.

Note that **add_sym()** will not load the newly created symbol into the system automatically. What it does is to create or to update the symbol into the specified symbol library.

add_tag

Create specific tag entity and append it to a given entity.

ENTITY **add_tag** (*ent,tag,text,mode,ps,angle,jflag*)

or

ENTITY **add_tag** (*id,tag,text,mode,ps,angle,jflag*)

- ENTITY** *ent* Entity handle of given entity, to which the new created tag is to append.
- int** *id* Always zero. If an integer value of zero is given as the first argument, *id*, the new created tag will be attached to the tag-list of drawing file level. Note that there is at most one and only one tag-list of drawing file level, with an ID number of zero.
- char** *tag[]* Character string of the tag name, of which only the first 11 characters are taken, and which will be converted to upper case.
- char** *text[]* Character string of the tag's initial content. The maximum length of this tag content is limited to 59 characters.
- int** *mode* Optional integer value specifying the initial status flag for the tag. Valid flags are **TAGDISP** (0x2000), specifying the tag content is viewable as text, and **TAGVAR** (0x4000), specifying the tag content is variable (can be modified by the operator). If this argument is missing, so as the rest, the initial mode will be **TAGVAR** only.
- POINT** *ps* Optional starting point of the text display of the tag content if it is displayed. If this argument is missing, the (0,0) point is assumed.
- double** *angle* Optional angle value in units of degree, specifying the text display angle of the tag content if it is displayed. If this argument is missing, a zero degree is assumed.
- int** *jflag* Optional integer value specifying the text justification flag for the text display of the tag content if it is displayed. If this argument is missing, then the baseline-left justification will be assumed for the text by default.
- The text justification flag consists of two parts: the lower nibble (bit 0 to bit 3) for the horizontal adjustment, and the next nibble (bit 4 to bit 7) for the vertical adjustment, as described below:

Bit 0-3:	Horizontal adjustment
0	Left side justification
1	Center justification
2	Right side justification
else	Reserved, Left side justification at present
Bit 4-7:	Vertical adjustment
0	Base-line justification
1	Bottom justification

- 2** Middle justification
- 3** Top justification
- 4** Middle height justification
- else** Reserved, treated as base-line justification.

Return Entity handle of the newly created tag entity, which is appended after the given entity.

This function appends tag entity to the end of a tag-list of a given entity. A tag-list is a linked list of tag entities appended to a geometry entity. If the given entity is also a tag, it must be part of a tag-list. So, the new one will be appended to the end of the list.

Note that this function will not check the redundancy of the newly created tag entity against the tag-list. It merely append the new one after it. So, it is possible to append tag entities of the same tag name to the same tag-list. To check if a specific tag is already in a tag-list, use the **srch_tag()** function. To modify an existing tag's content and/or attributes, or even the tag name, use **ent_read()** and **ent_write()** in the read-modify-write manner. To travel along a tag-list, starting from the given entity, use the **ent_tnext()** function. See also respective sections.

TwinCAD supports tag attributes in work drawing file level, which attach to no entities but the whole drawing. Such tag attributes will become private tag attributes when the whole drawing is loaded as a block. These tag attributes are useful and essential in auto-management of work drawing, since utility programs may directly access predefined tag attributes and determine the management status of the drawing.

However, there is no direct command support, currently, for users to create the tag attributes of drawing file level. But, with the TCL interface functions supported, should the operator need to create such tags without the decent TCL programming, he may directly issue the TCL functions from the command line, by supplying the arguments explicitly.

asin

Intrinsic function to calculate the trigonometric arc sine function of the argument value in units of degree.

double asin (*val*)

double val Sine value lies within -1. and 1.

Return The angle value in units of degree, which lies between -90° and 90°.

Note that the result of this function is in units of degree. It is the **asinr()** that returns the result in radians. This is different from the one in the standard C library. If the argument is not in the valid range from -1. to 1., it will cause a run-time error: *Operand Value Over Range*.

asinr

Intrinsic function to calculate the trigonometric arc sine function of the argument value in units of radian.

double asinr (*val*)

double val Sine value lies within -1. and 1.

Return The angle value in units of radian, which lies between $-\pi/2$ and $\pi/2$. See also **asin()**.

atan

Intrinsic function to calculate the trigonometric arc tangent function of the argument value in units of degree.

double atan (val)

double val Tangent value.

Return The angle value in units of degree, which lies between -90° and 90° .

Note that the result of this function is in units of degree. It is the **atanr()** that returns the result in radians. This is different from the one in the standard C library.

atan2

Intrinsic function to calculate the trigonometric arc tangent function of the argument value in the form dY/dX in units of degree.

double atan2 (dy,dx)

double dy, dx The delta X and delta Y component of a vector, of which the angle direction is to return. The signs of the dx and dy determine the quadrant for the returned angle. The result is an angle between the positive X-axis and a line drawn from the origin through the point (dx,dy) .

Return The angle value in units of degree, which lies between 0° and 360° . If the value of dx is zero, the result will be either 90° or 270° , depending on the sign of the dy . If both dx and dy are zero, then it return zero.

Note that the result of this function is in units of degree. It is the **atan2r()** that returns the result in radians. This is different from the one in the standard C library.

A macro to return the direction angle of two points can be given as:

```
#define ppangle(ps,pe) atan2((pe).y-(ps).y,(pe).x-(ps).x)
```

And also the direction angle of a line, using the above macro:

```
#define lnangle(ln) ppangle((ln).ps,(ln).pe)
```

atan2r

Intrinsic function to calculate the trigonometric arc tangent function of the argument value in the form dY/dX in units of radians.

double atan2r (dy,dx)

double dy, dx The delta X and delta Y component of a vector, of which the angle direction is to return. The signs of the dx and dy determine the quadrant for the returned angle. The result is an angle between the positive X-axis and a line drawn from the origin through the point (dx,dy) .

Return The angle value in units of radians, which lies between 0 and 2π . If the value of dx is zero, the result will be either $\pi/2$ or $3\pi/2$, depending on the sign of the dy . If both dx and dy are zero, then it return zero. See also **atan2()**.

atanr

Intrinsic function to calculate the trigonometric arc tangent function of the argument value in units of radian.

double atanr (*val*)

double val Tangent value.

Return The angle value in units of radian, which lies between $-\pi/2$ and $\pi/2$. See also **atan()**.

beep

Make a sound of beep!

void beep()

block

Add or update a specified block definition to the drawing database.

ENTITY block (*blkname, insbase, entlst*)

char blkname[] Name of the block to create or update.

POINT insbase Insertion base point relative to those selected entities.

SLIST entlst Selection list of entities that define the drawing block.

Return Entity handle pointing to the newly created BLOCK entity. Null handle is returned if the operation fail due to bad block name used.

This function call is almost equivalent to the function call `command("BLOCK...")`, except that it will create the block immediately without asking whether to make an insert or not, nor will it enter the user interface to help the user to create tag attributes.

To add tag attribute definition to the block, use the Text entities to define the tag attributes and include them in the selection list.

For a Text entity to be converted into equivalent Tag attributes definition, it must contain the text in the format:

&&{!#} tag-name { : initial text string }

where

&& Fixed identification codes to indicate that this is a tag attribute specification rather than an ordinary TEXT entity. The text content

	must start with these two characters so that the system will check and parse its content as a tag attribute definition.
!	Optional character to specify that the tag attribute is invisible. If this character is missing, the tag attribute will be visible by default.
#	Optional character to specify that the tag attribute is a constant attribute. If this character is missing, the tag attribute will be variable by default. Note that it does not matter whether this character is given after the optional '!' code or before it.
tag-name	Name of the tags. Only the first 11 characters are used.
:	Optional character to specify that an initial default text content for that tag attribute is followed (after the ':' code).

c or C

Intrinsic function to return a circle by geometry definition.

CIRCLE C (*argument...*)

Var argument Variable number of arguments of predefined type. Different number of arguments of different type will specify different definition of a circle.

Return Circle of the definition results.

There are many ways to define a circle. To provide each of them with separate functions is cumbersome in these functions's naming as well as in referencing them. Recalling that the operator may directly enter a valid TCL expression to the command line and utilize the power of TCL expression to facilitate his drafting operation, lengthy naming would certainly be unacceptable for most commonly used functions. This is one of the major reason to provide a general geometry circle definition function of **C()**, used as a Type Casting Function.

The methods used to define a circle depends on the arguments passed to the function, and are described below:

ent	Define a circle by directly loading it from an entity handle that points to an arc or a circle, or copying from an existing arc or circle. The argument must be of type ARC, CIRCLE, or ENTITY.
xc,yc,radius	Define a circle by giving its center coordinate values and radius value. All the three arguments are of double type.
p1,p2,p3	Define a circle that passes through the given three points. All the three arguments must be of POINT type. This function is the same as that of c_ppp() .
pc,radius	Define a circle with given center position and radius value. The first argument is a point, while the second one a double value.
pc,pnt	Define a circle with a given center position (the first argument) and a point it passes through (second argument). Both the arguments are of POINT type.
pc,ln	Define a circle with a given center position (the first argument) and tangent to a given line (the second argument). The first argument is of POINT type and the second, LINE type. The second argument may be an entity handle that points to a line in the drawing database.

Note that this intrinsic function will be invoked automatically by the pure assignment operator (=) when the data type of the L-value is CIRCLE. For example, the expression

```
C1 = 10,20,5
```

will be equivalent to the expression

```
C1 = C(10,20,5)
```

where the **C()** function is called by the assignment operator to evaluate the comma expressions to the right of it, which define a circle at position (10,20) with a radius of 5.

c_pccs

Find circles passing through a point and tangent to two circles.

int **c_pccs** (*p1,c2,c3,result*)

POINT *p1* Point through which the resulting circles must pass.

CIRCLE *c2, c3* Circles to which the resulting circles are tangent.

CIRCLE *result[]* Array of circles where the answers shall be returned.

Return Integer, number of resulting circles. If it is 0, it means there is no such geometry answer.

This function calculates the possible geometry answers to the circles passing through a given point and tangent to two given circles. There are at most 4 possible answers.

c_plcs

Find circles passing through a point, tangent to both a line and a circle.

int **c_plcs** (*p1,ln2,c3,result*)

POINT *p1* Point through which the resulting circles must pass.

LINE *ln2* Line to which the resulting circles are tangent.

CIRCLE *c3* Circle to which the resulting circles are tangent.

CIRCLE *result[]* Array of circles where the answers shall be returned.

Return Integer, number of resulting circles. If it is 0, it means there is no such geometry answer.

This function calculates the possible geometry answers to the circles passing through a given point and tangent to a given line and a given circle. There are at most 4 possible answers.

c_plls

Find circles passing through a point and tangent to two lines.

int **c_plls** (*p1,ln2,ln3,result*)

- POINT *p1*** Point through which the resulting circles must pass.
- LINE *ln2,ln3*** Lines to which the resulting circles are tangent.
- CIRCLE *result[]*** Array of circles where the answers shall be returned.
- Return** Integer, number of resulting circles. If it is 0, it means there is no such geometry answer.

This function calculates the possible geometry answers to the circles passing through a given point and tangent to two given lines. There are at most 2 possible answers.

c_ppa

Find the circle passing through two given points which span a given include angle counter-clockwise with respect to the circle.

int **c_ppa** (*p1,p2,angle,result*)

- POINT *p1,p2*** Points through which the resulting circles must pass.
- double *angle*** Double angle value in units of degree.
- CIRCLE **result*** Pointer to a circle where the answer is to be returned.
- Return** Integer value, 1 if the answer is returned, 0, if there is no answer.

If the given two points coincides or the include angle is zero, then there will be no answer. The include angle is taken counter-clockwise from the first point to the second point.

c_ppcs

Find circles passing through two given points and tangent to a circle.

int **c_ppcs** (*p1,p2,c3,result*)

- POINT *p1,p2*** Points through which the resulting circles must pass.
- CIRCLE *c3*** Circle to which the resulting circles are tangent.
- CIRCLE *result[]*** Array of circles where the answers shall be returned.
- Return** Integer, number of resulting circles. If it is 0, it means there is no such geometry answer.

This function calculates the possible geometry answers to the circles passing through two given points and tangent to a given circle. There are at most 2 possible answers.

c_ppls

Find circles passing through two given points and tangent to a line.

int **c_ppls** (*p1,p2,ln3,result*)

- POINT *p1,p2*** Points through which the resulting circles must pass.
- LINE *ln3*** Line to which the resulting circles are tangent.
- CIRCLE *result[]*** Array of circles where the answers shall be returned.
- Return** Integer, number of resulting circles. If it is 0, it means no such geometry answer.

This function calculates the possible geometry answers to the circles passing through two given points and tangent to a given line. There are at most 2 possible answers.

c_ppp

Find circles passing through three given points.

int **c_ppp** (*p1,p2,p3,result*)

- POINT *p1,p2,p3*** Points through which the resulting circles must pass.
- CIRCLE **result*** Pointer to the circle where the answer is to be returned, since there is at most one answer.
- Return** Integer, number of resulting circles. If it is 0, it means there is no such geometry answer.

This function calculates the possible geometry answers to the circles passing through three given points. There is at most 1 possible answer. The use of this function is preferred than the use of **c()** or **C()** function in that this function will return value indicating whether the given three points actually on a circle and not on a line.

c_ppr

Find the circles passing through two given points and with a given radius.

int **c_ppr** (*p1,p2,radius,result*)

- POINT *p1,p2*** Points through which the resulting circles must pass.
- double *radius*** Radius of the circles (absolute value is taken).
- CIRCLE *result[]*** Array of circles to hold the returned answers.
- Return** Integer value indicating the number of answer is returned. If it is zero, then no answer is returned. There are at most 2 answers.

If the given two points coincides or the radius value is smaller than half the distance between the two points, there will be no answer. If the radius value is exactly half of the distance value, there will be only one answer. For the other cases, there are at most 2 answers.

card_mid

Obtain the hardware **KeyCard**'s master ID number.

long card_mid()

Return Long integer value read from the installed **KeyCard**'s master ID number. If the returned value is zero, then the **KeyCard** is not installed. In this case, the running program may be **KeyPro**-protected version (using a hardware lock from the printer port).

card_zid

Obtain the hardware **KeyCard**'s zone ID number.

int card_zid()

Return Integer value read from the installed **KeyCard**'s zone ID number. If the returned value is zero, then the **KeyCard** is not installed. In this case, the running program may be **KeyPro**-protected version (using a hardware lock from the printer port).

Most of the **KeyCard**'s zone ID number will be the same for the products shelled in the same country. To identify the unique authorization, check the master ID number.

cb_dump Win

Dump the specific clipboard data to disk file.

long cb_dump (*fmid*, *fname*)

int *fmid* Clipboard data format ID to query and dump.

char *fname*[] Optional filename to dump the clipboard data of the specific format ID to disk file, if the data is available.

Return 0, if the clipboard data is not available; otherwise, the size of the data.

Use this function to retrieve the clipboard data to disk file, so that the TCL application may access its content via the file I/O operation.

cb_empty Win

Empty the Windows Clipboard.

void cb_empty()

Use this function to empty the Windows clipboard and release the memory used by the clipboard.

cb_format Win

Register/Get clipboard data format ID.

int cb_format (*name*)

char name[] Name of the clipboard data format to register for or to query about its ID.

Return Clipboard data format ID assigned by Windows. If it is not successful, it returns 0.

This function is used to register a customer clipboard data format by name and to obtain a format ID from the Windows, which will be used in subsequent clipboard data I/O. If a clipboard data format of the same name has been registered in Windows, it returns its assigned ID. Windows assigns the customer clipboard data format ID in the ranges 0xc000 to 0xffff.

TwinCAD will register a private clipboard data format named as "TCAMWrkFile" for Copying/Cutting/Pasting of drawing entities. The TCL application may use this function to obtain its format ID assigned by Windows and use it to query the clipboard to see if the specific data is available or not by the **cb_query()** function call.

TCL application may design its own clipboard data format and register its name to the Windows Clipboard Manager, and use the clipboard to pass data between different sessions of the application or even between different instances of TwinCAD.

cb_getdata Win

Directly read the clipboard data to data variables.

long cb_getdata (fmtid, data)

int fmtid Clipboard data format ID.

Var *data Multiple arguments of varies data type to receive the clipboard data, must be of array type or in pointer expression.

Return Same as **cb_setdata()**.

This function is used to retrieve data from the Windows clipboard directly.

If the clipboard data format is CF_TEXT or CF_OEMTEXT, **cb_getdata()** requires all the arguments be of string type only; otherwise, error will happen and the program be stopped. **cb_getdata()** will convert each text line from the clipboard into string and store them into the string argument in the given order.

If the clipboard data format is not CF_TEXT nor CF_OEMTEXT, **cb_getdata()** will process those arguments in the same way as the **fget()** does, except that the input will be directly from the clipboard.

See **cb_query()** for a list of pre-defined clipboard data format IDs.

All data stored in clipboard are owned by the Windows clipboard manager, and can be globally accessed by all Windows applications. However, when the clipboard is opened by one Windows application for data access, it can not be accessed by any other Windows applications until it is closed again. If any Windows application opens the clipboard and fails to close it, it will remain opened and always be an in-use state and no one else can access it any more until Windows is restarted.

So, for safety reason, TCL runtime does not allow TCL application to handle the opening and closing of the Windows clipboard explicitly. All access to the clipboard must be done in one single function call, to ensure that the clipboard will be opened and then closed properly.

For clipboard data size too large or format too complex, the TCL application may use **cb_dump()** first to create a disk image of it and then use ordinary file I/O to access it. Of course, the TCL application may declare a large array to retrieve the data and then access the array elements.

cb_load Win

Load external data file into clipboard.

int **cb_load** (*fmtid, name*)

- int *fmtid*** Clipboard data format ID to set with the content of the external file.
- char *name*** Name of the file of which the content will be loaded to the clipboard under the given format ID.
- Return** Size of the data loaded to Windows clipboard if successful; otherwise, it must be one of the error code given below:
- 0** Error, if the file is not found, or the clipboard is not available (currently in use by other application).
 - 1** Error, if TwinCAD is unable to allocate enough global memory from Windows to hold the whole content of the file, the file is too large or TwinCAD is unable to open the file.
 - 2** Error, if TwinCAD is unable to set the data to clipboard, mostly due to undefined clipboard data format ID.

This is a reverse operation of the **cb_dump()**. TCL application may create a data file of specific format, and use this function to send it to the Windows clipboard.

cb_query Win

Query the availability of Windows clipboard data.

int **cb_query** (*fmtid, id, name*)

- int *fmtid*** Clipboard data format ID to query. If it is 0, then it query all the available data format IDs from the clipboard. Optional, default to 0.
- int *id[]*** Optional integer array to hold all the available data format IDs from the clipboard, which are returned as the query result, effective only when *fmtid* is 0. If this argument is not given, only the number of available data format is returned. If the array size is not enough to hold the query result, exceeding informations will be dropped off.
- char *name[][]*** Optional string array to hold the format name of these available IDs returned as the query result, effective only when *fmtid* is 0. The returned informations will be truncated if the array size is not enough to hold it as a whole.
- Return** True if the specific format ID is available currently in clipboard, and false, if not. If the given format ID is 0, it returns the number of available data format IDs currently in clipboard. If the optional arguments are given, additional informations about these data format will also be returned.

Use this function to determine if the clipboard contains a data in a specific format. For private format, use **cb_format()** to get a format ID. The following is a list of Windows clipboard data format in macro definition form:

```
#define CF_TEXT          1
#define CF_BITMAP        2
#define CF_METAFILEPICT 3
#define CF_SYLK          4
#define CF_DIF           5
#define CF_TIFF          6
#define CF_OEMTEXT       7
#define CF_DIB           8
#define CF_PALETTE       9
#define CF_PENDATA      10
#define CF_RIFF          11
#define CF_WAVE          12
#define CF_OWNERDISPLAY 0x0080
#define CF_DSPTEXT      0x0081
#define CF_DSPBITMAP    0x0082
#define CF_DSPMETAFILEPICT 0x0083
```

See appropriate Windows document for details of these data format.

cb_setdata Win

Directly set data to clipboard in specific format

long **cb_setdata** (*fmtid*, *Var data...*)

int *fmtid* Clipboard data format ID to set with new data from the rest of the arguments.

Var *data* Multiple arguments of varies data type to set to the clipboard.

Return Size of the data loaded to Windows clipboard if successful; otherwise, it must be one of the error code given below:

- 0** Error, if no valid data to set to the clipboard, or the clipboard is not available (currently in use by other application).
- 1** Error, if TwinCAD is unable to allocate enough global memory from Windows to hold the whole content of the data.
- 2** Error, if TwinCAD is unable to set the data to clipboard, mostly due to undefined clipboard data format ID.

This function is used to set data to the Windows clipboard directly.

If the clipboard data format is CF_TEXT or CF_OEMTEXT, **cb_setdata()** will process those arguments of string type only. Each string will be output as a text line terminated by a <CR/LF> pair, and null strings will be output as empty lines.

If the clipboard data format is not CF_TEXT nor CF_OEMTEXT, **cb_setdata()** will process those arguments the same as the **fput()** does, except that the output will be directed to the Windows clipboard.

See **cb_query()** for a list of pre-defined clipboard format ID.

cc_pos

Determine the relationship between two given circles.

int **cc_pos** (*c1,c2*)

CIRCLE *c1,c2* Circles between which the relationship is to be checked.

Return Integer value representing the relationship between the circle C1 and the circle C2, as listed below:

- 0** Either C1 contains C2 or C2 contains C1, no intersections.
- 1** Both circles are tangent to each other, while either one is contained in another.
- 2** Both circles intersect with each other and either one's center is contained in another.
- 3** Both circles intersect with each other and neither one's center is contained in another.
- 4** Both circles are tangent to each other, while neither one contains another.
- 5** Both circles are apart from each other, no intersections.

If both given circles are the same (same centers and same radius), a 1 is assigned for this trivial case.

ceil

Intrinsic function to round the argument up to an integer (toward the positive infinity).

double **ceil** (*value*)

double *value* Double argument to find the ceil value.

Return Double value that is the smallest value not less than the argument and that is also an exact mathematical integer. If the value of the argument is already a mathematical integer, then the result equals to the argument.

centroid

Calculate the centroid coordinate of an area bounded by a close profile, or of a wire.

POINT **centroid** (*Argument...*)

Var *argument* Variable number of arguments of predefined type. Different number of arguments of different type will specify different calculations.

Return Point of the calculated Centroid Coordinate.

Different arguments passed to the function will have different processing, as described below:

- ent,type** The first argument is an entity handle that points to a circle, ellipse or a closed polyline. The second argument, *type*, is an integer of either 0 or 1 and is optional. If the value of *type* is 0, the function calculates the centroid of the closed area; otherwise, it returns the centroid of the wire. Note that if the first argument is an open polyline, the centroid of the area will assume that it is closed by a line from its end point to its start point.
- p1,p2,p3** Calculate the centroid of the area bounded by the triangle defined by three points. The three arguments must be of POINT data type or entity handles pointing to POINTs.
- p1,a2** Calculate the centroid of the area bounded by a line from a point *p1* to the start point of an arc *a2*, along the arc to its end point, and back to the point *p1*. The second argument can also be an elliptic arc in ELLIPSE data type.
- p1,l2** Calculate the centroid of the area bounded by a point *p1* and a line *l2*.
- a1** Calculate the centroid of the wire given by an arc *a1*.
- L1** Calculate the centroid of the wire given by a line *L1*.

chmod

Change the attribute of a given file.

int **chmod** (*filename,mode*)

char filename[] String, name of the file of which the attribute is to be changed.

int mode New attribute value of file to change to. The attribute of a file that can be changed are listed below:

- Bit 0** 1, if Read-only file, 0, if not
- Bit 1** 1, if Hidden file, 0, if not
- Bit 2** 1, if System file, 0, if not
- Bit 5** 1, if file archived, 0, if not
- Else** All else bits are ineffective.

Return 0, if the operation has succeeded; otherwise, error code from DOS, as given below:

- 2** File not found
- 3** Path not found
- 5** Access denied

The equivalent DOS command is ATTRIB (external command). This function is useful for TCL application to change an existing file's attribute to Read-Only mode or Hidden, so that it will not be erased nor to be seen by the operator.

clayer

Obtain layer information from system.

LAYERclayer (*name*)

char *name*[] String, optional, name of layer information to retrieve. Name string will be converted to upper case.

Return If the argument *name* is omitted or it is a null string, the information of the current layer is returned. Otherwise, the layer of the same name will be searched for and returned if it is found. If the layer of the specific name is not found, the function will return the first layer entry in the system table, so that the subsequent information of the layer can be retrieved via the **tbl_next()** function calls. To check whether the specific layer exists or not, further checking on the returned information is required.

This function is used to retrieve layer information only. The calling of this function will not change the current layer anyway. Use the **layer()** function to switch layers, create new layers, and modify the property of an existing layer. Note that the layer's name must be in capital characters, and only alphanumeric characters, '\$' (dollar) and '_' (underscore) characters are accepted in the name string.

Example: To obtain current layer

```
LAYER    curlayer;
curlayer=clayer();
printf("\nCurrent layer: %s, Ltype: %s, Color: %d",
      curlayer.name, curlayer.ltype, curlayer.color);
• To test if a layer is present or not
if ( clayer("BLUE").name=="BLUE" )
    printf("\n'BLUE' layer is present");
else
    printf("\n'BLUE' layer is not present");
```

clip_arc

Clip an arc by a window box.

int clip_arc(*arc*, *pMin*, *pMax*, *ares*)

ARC *arc* Arc to be clipped.

POINT *pMin* Minimum corner point of the clipping window.

POINT *pMax* Maximum corner point of the clipping window.

ARC *ares*[] Resulted arcs after clipping.

Return Integer value indicating the clipping state of the arc:

- 1 The arc is totally invisible from the clipping window.
- 0 The arc is totally visible in the clipping window. No clipped arcs are returned.

- 1 The arc is partially visible, and one clipped arc is returned.
- 2 The arc is partially visible, and two clipped arcs are returned.
- 3 The arc is partially visible, and three clipped arcs are returned.
- 4 The arc is partially visible, and four clipped arcs are returned.

clip_line

Clip a line by a window box.

int clip_line(*ln, pMin, pMax, lres*)

LINE *ln* Line to be clipped.

POINT *pMin* Minimum corner point of the clipping window.

POINT *pMax* Maximum corner point of the clipping window.

LINE **lres* Resulted line after clipping.

Return Integer value indicating the clipping state of the line:

- 1 The line is totally invisible from the clipping window.
- 0 The line is totally visible in the clipping window.
- 1 The line is partially visible, and .ps is clipped.
- 2 The line is partially visible, and .pe is clipped.
- 3 The line is partially visible, and both .ps and .pe are clipped.

codetype Win

Return the possible language of a specific code.

int codetype (*code*)

or

int codetype (*text*)

int *code* A single code to check.

char *text[]* Code string to check.

Return Integer bit flag indicating the possible language type of the given argument value:

- Bit 0:** 1, the code/string can be BIG5 coded.
- Bit 1:** 1, the code/string can be GB2312 coded.
- Bit 2:** 1, the code/string can be SJIS coded.
- Bit 3:** 1, the code/string can be Korean coded.
- Bit 7:** 1, the code can be pure 7-bits Ascii coded.

This function is used to analyze a given text string or a single code to see what kind of language it is possibly from.

com_close

Close a specific communication channel.

void **com_close** (*cdr*)

int *cdr* Channel descriptor from 0 to 3, corresponding to device COM1: to COM4: respectively.

If the specific communication channel has been opened, this function will close it. The memory buffer allocated for the channel will be released. The receiving interrupt on the chip will be disabled. The status of the 8259 interrupt controller will be restored and the channel service handler will be replaced by the previous one before the channel is opened by **com_open()**.

com_getc

Test and read a code from receiving buffer or from the chip directly.

int **com_getc** (*cdr*)

int *cdr* Channel descriptor from 0 to 3, corresponding to device COM1: to COM4: respectively, or the base port address of the communication chip. The address must be greater than 0xff.

Return 0, if the data is not available; otherwise, the data read.

If the first argument, *cdr*, is given within 0-3, the receiving queue buffer will be checked for the data input. It return 0 if no data available; otherwise, it return the next data read from the buffer. Note that data 00 will be returned as 0x100.

If the *cdr* is given larger than 0xff, it is then taken as the base address of the communication chip. A direct polling read of the chip register will take place. If a data is available from the chip's latch buffer, this function will return the LSR status at the high byte, and low byte the character read. Otherwise, it returns 0.

Care must be taken if a direct port address is used for polling at the chip. The system can not verify if the address is correct or not. It may jeopardize the system if the port address is not correct!

com_open

Open a specific communication channel for series I/O purpose.

int **com_open** (*cdr, rxsize, irqno*)

int *cdr* Channel descriptor from 0 to 3, corresponding to device COM1: to COM4: respectively.

int *rxsize* Optional data receiving buffer size from 64-4096. The default is 256. Value less than 64 will be set to 64, and larger than 4096 will be set to 4096.

int *irqno* Optional IRQ number used for the channel in DOS version. Valid values are 2, 3, 4 and 7. All else values will be ignored and the default will be used, as the table given below:

Cdr	COMn:	Port	IRQ#	INT#
0	COM1:	3f8h	4	0CH (Standard)
1	COM2:	2f8h	3	0BH (Standard)
2	COM3:	3e8h	4	0CH
3	COM4:	2e8h	3	0BH

Return 0, if the system fail to open the channel due to memory shortage, or else the actual size of the receiving buffer allocated.

This function provides an interrupt driven mechanism on data input from the communication port. A circular queue buffer is maintained for the incoming data. The TCL application may then use **com_getc()** function to retrieve the data from the queue buffer. The **com_close()** must be called to close the channel after the channel is not used.

At most 4 such communication channels can be setup. Each is identified by a number from 0 to 3. If the specific channel is already opened when this function is called, it will be closed first and then re-opened again with the new parameters.

The XON/XOFF protocol on data layer is also implemented by the queue buffer manager in the interrupt driven mechanism. It will send an XOFF code (DC3) when the available buffer is below 16 bytes, and send an XON code (DC1) to resume the data input when the available buffer size is larger than 32 bytes. If the external device does not implement the XON/XOFF protocol, then buffer over-run may occur when the data processing rate is slower then the data incoming rate. Such error can not be returned at the time being. The maximum data incoming rate is about 1/10 of the baudrate.

The system will automatically close all the opened I/O channels when

1. A TCL program exits (no checking on whether the channel was opened by it).
2. DOS shell is entered by ^D (DOS version only).
3. DOS command is entered so that the system will be swapped out (DOS version only)

The system can not resume the I/O channel automatically once it is closed upon the above conditions. The TCL application must re-open it again if necessary.

Windows Specific

TwinCAD does not handle the series port I/O directly in Windows. Instead, it calls Windows API for the job. So, the description in the previous paragraphs are modified as such:

- The channel descriptor, *cdr*, is from 0 to 8, corresponding to device COM1: to COM9: respectively. Direct port address I/O is not supported.
- The default size of the data receiving buffer is 2048 bytes and the smallest buffer size is 128 bytes. The maximum size is 32767 bytes.
- TwinCAD will allocate 512 bytes for the transmission buffer. The DOS version does not support the use of a transmission data buffer.
- The third argument, *irq*, is no longer used to specify the IRQ number and is default to 0. However, it is used for the following bit flag controls:

- Bit 8:** ON, disable XON/XOFF control in data transmission.
- Bit 9:** ON, disable XON/XOFF control in data receiving.
- Bit 10:** ON, disable CTS/RTS hardware hand-shaking.
- Bit 11:** ON, disable DTR/DSR hardware hand-shaking.
- Bit 15:** ON, disable closing the communication port automatically. The port must be closed explicitly by **com_close()**.
- Else:** Reserved, and should be 0.

com_putc

Send a single byte out through the communication channel.

int com_putc (*cdr,cc,check*)

- int *cdr*** Channel descriptor from 0 to 3, corresponding to device COM1: to COM4: respectively, or the base port address of the communication chip (DOS version only). The address must be greater than 0xff.
- char *cc*** Single byte data to be send through the channel or the port.
- int *check*** Optional argument specifying whether to check the CTS and DTR signal or not. 0 to disable the checking and non-zero to enable the checking. Default to 1. This feature is supported only after V3R3b in DOS version.
- Return** It returns 0 if the operation succeeds; otherwise, it returns the chip status with the time-out bit flag set in DOS version, or error flag in Windows version. see **com_state()** for details.

DOS Specific

This function implements the RTS/CTS single handshaking on the line. This handshaking can not be disabled unless the third optional argument is explicitly given as zero. If the external device does not handle RTS/CTS signal, the pin 4 and pin 5 must be shorted together.

Care must be taken if a direct port address is used for polling at the chip. The system can not verify if the address is correct or not. It may jeopardize the system if the port address is not correct!

Windows Specific

The channel descriptor, *cdr*, is from 0 to 8, corresponding to device COM1: to COM9: respectively. Direct port address I/O is not supported.

Whether the communication port I/O employs the RTS/CTS or DTR/DSR hardware handshaking is determined at **com_open()** function call. So, the third argument, *check*, is meaningless in Windows version.

As a 512-byte transmission buffer is used to transmit the data in background, this function will return quickly and will not wait for the code to be sent through the line. If the transmission buffer is full, it returns an error status with bit 8 set (0x100). In this case, the code is not placed in the transmission buffer yet, and the TCL application must re-send the code and poll the returned value until it returns 0. The TCL application must

examine the other bit flag for hardware I/O error, in case the data jam was caused by the hardware.

com_state

Return the communication device status.

int com_state (cdr)

int cdr Channel descriptor from 0 to 3, corresponding to device COM1: to COM4: respectively, or the base port address of the communication chip (DOS version only). The address must be greater than 0xff.

Return The status of the port.

DOS Specific

The status of the port returned is given below: (MSR in low byte and LSR in high byte)

- Bit 0:** Delta Clear To Send (DCTS).
- Bit 1:** Delta Data Set Ready (DDSR).
- Bit 2:** Trailing Edge Ring Indicator (TERI)
- Bit 3:** Delta Receive Line Signal Detect (DSLSD).
- Bit 4:** Clear To Send (CTS).
- Bit 5:** Data Set Ready (DSR).
- Bit 6:** Ring indicator (RI).
- Bit 7:** Received Line Signal Detect (RLSD).
- Bit 8:** Data Ready (DR).
- Bit 9:** Overrun Error (OR).
- Bit 10:** Parity Error (PE).
- Bit 11:** Framing Error (FE).
- Bit 12:** Break Interrupt Detect (BI).
- Bit 13:** Transmitter Holding Reg. Empty (THRE).
- Bit 14:** Transmitter Shift Reg. Empty (TSRE).
- Bit 15:** Device not ready, time-out error (added by **com_putc()**).

For detail explanation about the status bit flag, please refer to 8250 chip data sheet.

Windows Specific

The channel descriptor, *cdr*, is from 0 to 8, corresponding to device COM1: to COM9: respectively. Direct port address I/O is not supported.

The status of the port returned is given below:

Bit 0:	CE_RXOVER, receiving queue overflowed.
Bit 1:	CE_OVERRUN, character was not read from the hardware before the next character arrived. The character was lost.
Bit 2:	CE_RCPARITY, hardware detected a parity error.
Bit 3:	CE_FRAME, hardware detected a framing error.
Bit 4:	CE_BREAK, hardware detected a break condition.
Bit 5:	CE_CTSTO, CTS (clear-to-send) timeout.
Bit 6:	CE_DSRTO, DSR (data-set-ready) timeout.
Bit 7:	CE_RLSDTO, RLSD (receive-line-signal-detect) timeout.
Bit 8:	CE_TXFULL, transmission queue was full when a function attempted to queue a character.
else:	Reserved.

com_ugetc

Un-get (put) a single byte back to the receiving buffer.

int com_ugetc (*cdr,cc*)

int <i>cdr</i>	Channel descriptor from 0 to 3 in DOS version, from 0 to 8 in Windows version.
char <i>cc</i>	Single byte data to be put back to the queue buffer.
Return	It returns 0 if the channel is not opened and the queue is not available, or there is no room available from the queue buffer. Otherwise, it returns the current number of available buffer size in bytes for the new coming data.

command

Execute **TwinCAD** system commands by creating a temporary command scripts.

int command (*arglist...*)

Var <i>arglist</i>	At most 16 arguments can be given to this function call. Only arguments of string type, numerical value and POINT type are accepted. Numerical values (integer or floating point) and POINT type values are converted into equivalent strings. These strings are inserted to simulate the command script input to the system command prompt. Each argument is responsible for such command script input line.
Return	Number of undo sections has been created.

This function is used to create a temporary script and then submit it to the system kernel for execution as if the script was entered from the command line, or read from the menu

file. It provides a direct and easy interface between **TCL** programs and the **TwinCAD** system kernel. All the rules about the command script arguments are nearly the same as those for the lines from menu files and external command script files (SCR), with only a few exceptions:

- The semicolon ';' will not be converted into carriage return code as it is from the menu files. Use "\n" or "^M" instead.
- The dollar sign '\$' will not be taken as a part of sub-menu call syntax. There is no sub-menu call support in such a temporary script.
- The plus sign '+' at the end of a line will not be ignored or taken as a continuing indicator. It will be a part of the input.
- No extended commands (TCL programs loaded by CLOAD) can be executed from such temporary scripts. These command should be called directly as a function (in exact cases). This is because these loaded extended commands may hide the system's prime commands from being executed from the command line. With this restriction, it can be guaranteed that the commands invoked by **command()** will be the system prime commands.
- Null string from the argument will be replaced with " ^" which will generate a single space to the command line.
- Auto repetition of a prime command by means of a space or a carriage return code is ineffective. All prime commands must be explicitly given. This means that the leading spaces before a prime command will be ignored, while the trailing spaces will not cause the command to be executed automatically again. This will ease the TCL programmer in counting the spaces. Nevertheless, once a prime command is executed, all characters from the scripts are responsible to simulate the input to the command prompt until the command exits, including the spaces.

There is no limitation on how many prime commands can be called by the command scripts setup from the arguments; however, if the scripts passed by these arguments do not end an invoked prime command's execution thoroughly, the command in execution will turn to the operator asking for more input to end the command.

Currently, there is no explicit and easy way to verify if a Prime Command issued from the **command()** function call has worked correctly or not. There is no error trapping mechanism in the current version. However, if the TCL program does one thing at a time using **command()**, then the return number of undo sections created may serve as an indicator whether the command has succeeded in its operation or not.

Not all the prime commands can be successfully executed by means of this **command()** call, such are SCRIPT, RSCRIPT, RESUME, MULTIPLE, and REDO. Some commands may produce side effects and force the scripts to end prematurely, such are MENU, PURGE, and NEW/C. For example, the expression:

```
command("MENU ...", "SAVE...")
```

will not execute totally as what is expected by the programmer. The SAVE command will not be executed since the **MENU** command will re-initialize the menu system and empty the active menu script buffer.

There are also rules about supplying input to the command prompts that require a filename or a text string. These rules are the same as those for the menu scripts, and are so important to mention them here again:

❖ **Filename input**

which forces the system to evaluate the string and obtain the string as a string constant. The effect of the backslash code is hidden in the quoted string.

Executing TCL Program Using `command("RUN ...")`

It is allowed to execute another TCL program using the `command("RUN...")` function call with the following considerations:

- The file handles opened in the mother TCL program will be hidden from the child program, so are the text window handles.
- The maximum number of file can be opened by the child program is still 10 files, provided that there are enough file handles left in the DOS system.
- The total maximum number of graphic text window is 10. If the mother TCL program has used up all the graphic text windows, then the child TCL program will fail to open one for itself.
- The exiting of a child program will close all the resource opened for the child program (such as files, text windows), but this will not affect those for the mother TCL program.
- The use of application text generation file (see `tg_open()` and `tg_read()`) is global to all TCL programs. The system will not close or open it automatically. It is the TCL application's responsibility to maintain its fitness. A general guide line is given here: If an independent TCL application must use the TG file, it must save the current active TG filename before opening its own, and restore the original one before it exits.

CLOAD Another TCL Program Using `command("CLOAD ...")`

It is allowed to load in TCL programs using the `command("CLOAD ...")`. However, once this is done, the TCL program memory allocated by current running TCL program will not be released after the TCL has exited. It means that a part of the swapped disk space is taken and will never be used again, until the whole system exits. So, it is not recommended to load in TCL programs from within a large TCL applications.

Executing END/QUIT commands in `command("...")`

It is possible to execute the **END** and **QUIT** command to terminate the whole system. However, the system will not terminate immediately when the command is executed. A special flag, stored in the system variable `@SYSENDFLAG`, will be set and the execution will be continued until all active level of command shells or TCL applications have already terminated and the command level has returned to the top level of system command prompt.

Executing UNDO Command Using `command("U")`

It is possible to issue **UNDO** commands using `command("U")` to undo the previous `command()` function call or drawing database modification by the TCL runtime functions. See `set_undo()` function for further details on this subject.

A TCAD Temporary Command Shell

This is a special trick in using `command()` in TCL programming. You may create a TwinCAD temporary command shell by the function call:

```
command("\");
```

which tells the command shell to hold for user to input a command. Once the operator has entered a valid command, the command shell will continue to process the command. While there is no more script after the backslash ('\'), the command shell will turn its face to the man/machine interface for user input to complete the command. It returns control when the command is completed.

Address Cursor Pointer's Dragging Position in command()

Some of the Prime Commands rely on the user's interactive dragging operation to complete the job, where the dragging position of the cursor pointer may serve as some kind of indicator to determine how the job will be done.

An example of such is the **CIRCLE** command. It depends greatly on the interactive dragging operation to find the desired solution. But, if a TCL programmer wants to utilize the **CIRCLE** command to find a specific solution without resorting to the aids of the user from the interface, he will need to supply the additional cursor dragging position to the command processor. Otherwise, the command may fail to find the correct one for him, since a part of the solution factors are determined by the dragging position, which is handled by the command's dragging handler.

During the processing of a Prime Command, the data point coming from the menu script (as supplied by the **command()** function's arguments) will be read immediately by the system without calling the user interface. So, there will be no chance for the active dragging handler to obtain the cursor pointer position and therefore to make the required dynamic calculation which in turns becomes the solution factors.

A special *Cursor Snap Directive*: "**PTR**" is added to help the TCL programmer to address the cursor pointer position as if the user has moved the cursor pointer to that specific position in response to the command prompt. This special Snap Directive is intended for use with TCL programming. It would be nonsense for a user to input this snap directive interactively.

At receiving this "**PTR**" directive, the system will nest the input request down one level where a data point input is read from the input source. After obtaining the data point coordinates, it will address the cursor pointer to that location and then call the current active dragging handler with that coordinates twice; one for drawing the cursor and the other for erasing it. And then, it simply resumes the previous level of input request and continues the operation.

What it does is to give the interactive dragging handler a chance to fulfill its duty on the job, so that the command operation may work correctly when there is no real user interaction.

Again, take the **CIRCLE** command as an example, in which the dragging cursor position is an important solution indicator. So, the TCL programmer, knowing which solution is to find, must supply this additional information to the command processor via the use of "**PTR**" directive to address the cursor position, and thereby to indicate the choice of solution. An example command script is given like this:

```
command("CIRCLE TTR e1 e2 PTR p1 10 ");
```

assuming the e1 and e1 are pointing to valid entities and the pick points on them are also effective. The point p1 indicates the desired one from the 8 possible solutions.

Example: POINT vertex[5], center=0,0;

```
double   size=10.;  
for(i=0;i<5;i++)       // Calculate vertex of a star
```

```
vertex[i] = center + p(size,(90+i*72)A);  
command("line \@vertex[0] \@vertex[2] \@vertex[4]",  
        "\@vertex[1] \@vertex[3]","close");
```

copyfile

Copy a specific file from one pathname to another pathname.

long copyfile (*src,dest*)

char *src*[] The pathname of the source file.

char *dest*[] The pathname of the destination file.

Return Number of bytes being copied if it is successful, or one of the following error codes:

0 if nothing done (same file).

-1L if source file error.

-2L if destination file error.

Windows Specific

If the source file is a Windows compressed file, it will be decompressed during copying.

COS

Intrinsic function to calculate the trigonometric cosine function of the argument value, where the argument is taken to be in units of degree.

double cos (*angle*)

double *angle* Angle value in units of degree.

Return Cosine value of the given angle in double.

Note that the argument is in units of degree! It is the **cosr()** function that takes the angle in radians. This is different from the one in Standard C library. The reason for such implementation lies in the fact that the **cos()** function may be called in the expression from the command line input, where the operators are using the degree for angle specification.

COSR

Intrinsic function to calculate the trigonometric cosine function of the argument value, where the argument is taken to be in radians.

double cosr (*angle*)

double *angle* Angle value in units of radian.

Return Cosine value of the given angle in double.

or

POINT cosr (*z*)

POINT *z* Complex number in POINT data type.

Return Complex number as the Cosine result of the given complex number.

crc_check

Build CRC checking table and perform CRC checking.

int **crc_check** (*table, poly*)

int *table*[256] Integer array of 256 elements to return the CRC look-up table for subsequent CRC checking calculation.

int *poly* CRC polynomial constant. It can be 0xA001 for ANSI CRC-16 standard, 0x8408 for CCITT standard, or any other values of interest.

Return Meaningless.

This function call is used to build a 256-word CRC checking look-up table with a given CRC-polynomial constant. The table should be used in subsequent **crc_check()** calls for the CRC calculation.

To perform the CRC checking, call the function in the following way:

int **crc_check** (*table, crc, var data,...*)

int *table*[256] The CRC look-up table.

int *crc* Initial CRC value (0 for ANSI CRC-16, and 0xFFFF for CCITT), or the current CRC value for continuous CRC calculation.

Var *data* Variable number of arguments (maximum to 15) of accepted data types: POINT, LINE, ARC, CIRCLE, LTYPE, LAYER, and all elementary data types such as char, int, long, double, etc. Note that the ENTITY, ENTDATA and other data type not in the previous list will not be accepted as the arguments, since their sizes are varied and structural contents are subjected to change.

Return CRC value after checking the variable data with the given initial value.

This function call is used to calculate the CRC checking value upon variable data. The argument data given will follow the rules for those given to **fput()** functions.

cspline

Fit a cubic spline curve out of a given set of data points.

ENTITY **cspline** (*pnts, n, type, angs, ange*)

POINT *pnts*[] Array of data points to fit.

int *n* Number of data points in the array.

int type	Type of boundary conditions for the cubic spline: 0 Clamped with default clamping condition. This is the default. 1 Clamped with starting angle specification. 2 Clamped with ending angle specification. 3 Clamped with both the starting and ending angle specification. 4 Natural or relaxed condition. 5 Cyclic condition.
double ang s	Optional, the clamping angle at the start point, taken only when the type value is 1 or 3.
double angle	Optional, the clamping angle at the ending point, taken only when the type value is 2 or 3.
Return	Entity handle pointing to the resulted polyline. Null entity handle is returned if no curve is created (data point error condition).

With the use of this function, a TCL application needs not to create the POLYLINE with these data points first only to apply the CSPLINE command. This will save a lot of resource in time and space.

The current setting of SPLINESEG variable will affect the number of arc segments generated in approximating the spline curve.

If the bit 4 (0x10) of the type value is ON, then no real curve will be generated. However, the tangent vector on each given node point (the data point) will be returned in the data point array, overwriting the original data point values. In this case, the function will return integer type value (number of vectors), instead of ENTITY type. This is a very special call. The caller must save the original data points to somewhere else before doing such kind of call.

date

Obtain current system date information.

STRING date (*idate*)

int idate[4] Integer array of 4 elements to hold the returning information of current system date, which are in sequence the year (1980-2099), month (1-12), day (1-31) and weekday (0-6).

Return The system date in string as "Fri Aug 3, 1986".

If it is the system date in string of interest, use the intrinsic function **date\$()** instead. See also **date\$()**.

date\$

Intrinsic function to return current system date in string.

STRING date\$()

Return The system date in string as "Fri Aug 3, 1986".

You may directly reference the **date\$()** function as a data variable as **date\$**. This function can be called from the command line to show the current system time, as if it were a command.

db_append

Append a new record to the end of database.

int db_append (data,...)

Var data Optional number of initial data for the newly added record.

Return The current record number (of the appended record).

This function is used to add a new blank record to the end of the database. The newly added record will become current record. Optionally, the caller may store initial data to the new record directly with the optional data arguments when the blank record is created and added to the database.

db_close

Close an opened database file (DBF).

int db_close (dbfname)

char dbfname[] Path name of the database file (DBF file).

or

int db_close (id)

int id Work area ID of the database file.

Return 0, if the operation successes, -1, if the file is not opened or the work area ID is not valid.

This function is used to close the specific database file. It must be called to terminate the use of a database file, so that the system may write out the data held in the work area.

db_create

Create a new database file (DBF).

int db_create (dbfname,dbstruc,nfield)

char dbfname[] Path name of the database file to create.

char dbstruc[][] Structure of the database file. See description texts.

int nfield Optional, number of field to create.

Return -1, if the file can not be created; otherwise, the number of data fields created in the new database file is returned.

This function is used to create a new blank database file with the specified field structures for each data record. If no extension is explicitly given, then the "DBF" will be assumed for the database file to create. If the file already exists, it will be cleared and rebuilt.

The field structures for each data record are specified by a two-dimensional character array, which is taken as a one-dimensional array of string. Each string must contain a valid field structure specification in the foren7.5be,7

Return 1, if the current record is marked for deletion; 0, if the current record is not marked for deletion.

This function is used to test whether the current record is marked for deletion or not. You may use **db_recall()** to recall the current record from deletion.

db_fid

Obtain a specific field's index number.

int db_fid (*field*)

char *field*[] Name of the field.

Return -1, if the specific field is not found from the current selected database; otherwise, the field index number (starting from 0) is returned.

This function is used to identify a specific field by name from the current selected database. If the field is found, its field index number will be returned. The field index number is used to identify the field for the field data read/write access.

db_field

Obtain the field informations by index.

STRING db_field (*fid*)

int *fid* Field index number.

Return Data field specification string in the format as "*field-name,type,width,dp*", if the field index is valid to the current selected database. Null string is returned if the index is not valid (out of range). See **db_create()** for the descriptions to the field specification string.

This function is used to retrieve the field structures of the database file.

db_fieldno

Obtain the field number information from selected database.

int db_fieldno()

Return The number of data fields for each record in the current selected database. 0, if no database is being selected.

This functions is used to return the field number information of the current selected database.

db_get

Read specific field data from current record.

STRING db_get (*fid*)

int *fid* Field index number.

Return Null string if the field index is out off the valid range or no database file is selected. Otherwise, the content of the specified data field from current record is returned.

This function is used to read the specific data field of the current record from the current selected database. Note that all data values stored in the database file are in form of ASCII text string.

db_goto

Move the record pointer of current selected database.

long db_goto (*where*)

long *where* Absolute record number to which the current record pointer is moved.

Return The current record number. If the return number is less than the specified number, it is at the end of the file.

This function is used to move the record pointer to anywhere in the current selected database. If the argument is zero, it effectively move the record pointer to the top of the database. If the argument is -1, it will move the record pointer to the bottom record of the database. Note that the record number starts from 0.

db_insert

Insert a specific number of blank records.

int db_insert (*nrec*)

int *nrec* Optional number of blank record to insert, default is 1. See description texts.

Return Number of blank records inserted.

This function is used to insert blank records into the database after the current record or before it. If the argument is positive, blank records will be inserted after current record. If it is negative, blanks records will be inserted before the current record. If the argument is zero, one is assumed. After the insertion, the current record pointer will be updated to point to the first inserted blank record.

Note: Inserting a blank record into database is a very slow process, as the system must move all the records downward to make room for the new records. If possible, use **db_append()** to add new records at the end of the database.

db_locate

Locate a specific record with a keyword.

int db_locate (*fid,keyst,flag*)

int *fid* Field index number of the record to match with the keyword.

char *keyst* Keyword of record to search for.

int *flag* Optional control flag for the searching, default is 0. See description texts.

Return 1, if found and the current record pointer is updated. Or 0, if not found and the current record pointer is not moved.

This function is used to locate a specific record with a keyword to match with a specified data field content, starting from the current record inclusively. If a matching is found, the current record pointer will be updated and this function will return 1. Otherwise, this function will return 0, and the record pointer will not be changed.

The optional *flag* is used to control the way the function try to match the data field with the keyword. It is a bit flag control value. Valid bit flags are described below:

- Bit 0:** 1, to match the keyword as a sub-string, or 0, to require a full matching of the keyword.
- Bit 1:** 1, to use the keyword as a pattern to match with the data field (overriding Bit 0), or 0, not to use it as a pattern.
- Bit 2:** 1, to exclude the leading spaces from the data field before doing the match, or 0, no to exclude the leading spaces.
- Bit 3:** 1, to exclude the trailing spaces from the data field before doing the match, or 0, not to exclude the trailing spaces.
- Bit 4:** 1, to match also the deleted records, or 0, to exclude the deleted records.
- Else** Reserved and should be zero.

db_logic

Test the specific data field for logical result.

int db_logic (*fid*)

int *fid* Field index number.

Return 1, for true, 0, for false, -1, for data undetermined.

This function is used to test a logical data field and obtain its equivalent logical state. The function will read the first character of the data field and check to see if it is a 'Y', 'y', 'T' or 't' for True, or a 'N', 'n', 'F' or 'f' for False.

db_pack

Pack the current selected database file.

int db_pack()

Return Number of deleted records being packed off.

This function will pack the current selected database and remove those records being marked for deletion permanently. If there is any deleted record in the database, after the call to this function, the current record pointer will be moved to the top of the database.

db_put

Write data to specific data fields.

int db_put (fid, data,...)

int fid Field index number.

Var data Optional number of arguments to pass data values for the data fields.

Return Number of data bytes written out

This function is used to update the specified data field of the current record with the given data. If more than one argument data are given after the field index number, the successive data fields are updated accordingly.

Data values can be character string, integer or double. If an integer or a double value is given, it will be converted into equivalent character string before being stored into the data field.

db_recall

Recall the current record from being marked for deletion.

int db_recall()

Return -1, if no database file being selected; 0, if the record is not marked for deletion; 1, if the operation is successful.

This function is used to remove the deletion mark from the current record. You may use **db_delete()** function to mark current record for deletion, and use **db_deleted()** to test whether the current record is being marked for deletion.

db_reccount

Obtain the total record number from current selected database.

long db_reccount()

Return The total number of records contained in the current selected database.

This function is used to obtain the total number of records contained in the current selected database, including those being marked for deletion, without altering the current record pointer.

db_recno

Return the current record number.

long db_recno()

Return The current record number (position of the record pointer).

This function is used to obtain the current record number of the current selected database.

db_select

Select an Opened Database File for Subsequent Access.

long **db_select** (*dbfname*)

char dbfname[] Path name of the database file (DBF file).

or

long **db_select** (*id*)

int id Work area ID of the database file.

Return -1L, if the file is not in use (opened) or the work area is not in use. Otherwise, the current record pointer to the selected database is returned.

This function is used to select an opened database for the subsequent read/write access. Note that only one database file can be selected for read/write access at one time.

db_skip

Relatively move the current record pointer.

long **db_skip** (*nrec*)

long nrec Optional record number to move the record pointer relatively. Default is 1, if this argument is missing.

Return The current record number.

This function is used to move the record pointer forward or backward, relative to the current record position, for a specific number of records. If the argument is negative, the record pointer is moved backward. If it is positive, the record pointer is moved forward. If it is zero, the record pointer will not be moved, but the current record number is returned.

If no argument is given, it is equivalent to move the record pointer forward by 1.

db_sort

Sort the current selected database on specific data field.

int **db_sort** (*fid, tofile*)

char fid Field index number.

char tofile[] Optional output file to hold the sorted result.

Return 1, if the database is sorted as required; 0, if the database needs not to be sorted (empty); or -1, if the database can not be sorted.

This function is used to sort the current selected database on a given data field in ascending order. However, due to the limitation of the memory, only those databases with no more than 8192 records can be sorted by this function in current version.

If the optional output file is not given, the current database will be updated with the sorted result. Otherwise, the system will write the sorted result directly to the output file without affecting the content of the current database.

db_use

Open and Use Database File (DBF).

int db_use (*dbfname*)

char *dbfname*[] Path name of the database file (DBF file).

Return The work area ID number (from 0 to 3) assigned for the DBF file, if it is opened and used successfully. If it is less than zero, it returns error:

- 1 If the file is not found.
- 2 If no more work area available for the file.
- 3 If the file is not recognized as a valid database file.

This function is used to open an existing database file and register its use to the system. The system will check and read its file structure for the subsequent read/write access to it. If no extension is given explicitly, the system will assume it as "DBF" automatically. If the file can not be found from the path specified, the current directory path and the directory paths specified in the TCADPATH environment variable will be searched through for it.

If the function succeeds in opening the database file, the file will become the current database file. Its current record pointer will be set to the first record in the file. If the database file has been opened (by previous **db_use()** function calls) when this function is called, nothing harm will be done. But, it will be selected as the current database file and the record pointer is moved to the top of the file, and its work area ID number is returned.

Currently, you can open and use at most 4 database files simultaneously, although the dBASE allows up to 10 files.

db_value

Read the specific data field for numerical value.

double db_value (*fid*)

int *fid* Field index number.

Return The evaluated value of the numerical data field.

This function is used to obtain the equivalent numerical value from the specific data field, assuming it is a numerical field. If the numerical string is delimited by commas, the function will remove the commas before evaluating it.

db_zap

Clear the current selected database.

int db_zap()

Return 1 if the operation is successful, 0 if not.

This function will remove all the records from the current selected database and leave the database structure intact. Note that this will permanently empty the current database file.

dbl

Intrinsic function to explicitly convert an integer to double.

double dbl (ival)

int ival Integer argument to convert to double.

Return Equivalent double value.

Note that this function is mainly used to force the evaluation of an expression or sub-expression be done in floating arithmetic, instead of integer arithmetic.

dde_close Win

Disconnect a DDE linkage and close an opened DDE channel.

int dde_close (chid)

int chid DDE channel ID returned by **dde_open()**.

Return True, if the DDE channel was opened and is closed now. False, if the channel ID is incorrect or the channel was not opened.

This function is used to close a DDE channel opened by the **dde_open()** function call. The DDE linkage with the external server will be disconnected and all the data memory associated with the channel will be freed.

dde_closeall Win

Disconnect all the DDE linkages and close all opened DDE channels.

int dde_closeall ()

Return Number of channels being closed.

int *chid* DDE channel ID returned by **dde_open()**.

char *cmdstr[]* Command string to execute.

long *wait* Optional timeout constant to wait for the server if server is busy. Default to 0.

Return True, if the command is sent successfully. False, if the channel ID is incorrect, or the channel is not opened, or the server does not response and cause a time-out.

This function is used to issue application command through the DDE linkage. Check the targeted application's documentation for the syntax and meaning of the supported commands.

Example: The pm_command() may be defined by the following program fragment:

```
int pm_command(char cmdstr[])
{
    int nPm, nRet;
    nPm = dde_open("Progman", "Progman");
    if ( nPm > 0 )
        nRet = dde_execute(nPm, cmdstr);
        dde_close(nPm);
        return nRet;
    }
return 0;
}
```

dde_hotlink Win

Establish a hot link on specific data item.

int dde_hotlink (*chid, item, format*)

int *chid* DDE channel ID returned by **dde_open()**.

char *item[]* Name of the data item to receive via hot link.

int *format* Optional data format ID. If this argument is not given, CF_TEXT format is assumed. See **cb_format()** for obtaining the customer data format ID.

Return True, if the operation is successful; False, if it fails.

This function is used to establish a hot link with the server on a specific data item. If it is successful, the server will automatically advise the client with the data whenever the data item has changed on the server side. The data will be latched in the channel buffer and the TCL application may use **dde_get()** and **dde_getline()** to read it. The data will be freed automatically whenever the the data is read over the end.

To stop a hot link, use **dde_coldlink()**.

dde_get Win

Read the latched data from specific DDE channel.

int dde_get (*chid, args,...*)

int *chid* DDE channel ID returned by **dde_open()**.

Var* *args* Variable number of pointer arguments (maximum to 15) of accepted data types: POINT, LINE, ARC, CIRCLE, LTYPE, LAYER, and all elementary data types such as char, int, long, double, etc. Note that the ENTITY, ENTDATA and other data type not in the previous list will not be accepted as the arguments, since their sizes are varied and structural contents are subjected to change.

Return -1, if end of data is encountered or no data to read, else the number of bytes read in.

This function is used to read the data from the specific DDE channel, latched by the latest **dde_request()** function call or by the latest received data through hot link. The reading will automatically update the data read position.

If the data was received via hot link (not requested by the explicit **dde_request()**), then reading over the end of the data will cause the data be discarded (freed) automatically. However, if the data was latched through **dde_request()** function call (under either cold or warm link), it will not be discarded automatically. You may use **dde_offset()** to move the data read position so as to read the data again, or to remove the data explicitly.

dde_getline Win

Read the latched data from specific DDE channel.

STRING dde_getline (*chid, tab*)

int *chid* DDE channel ID returned by **dde_open()**.

int *tab* Optional control flag, TRUE, if the TAB character will be taken as a line delimiter, and FALSE, it will be taken as ordinary data. Some DDE servers, such as EXCEL, use TAB character as the string delimiter in the item data. The default is false, which means to take LF as the line delimiter only.

Return The next text string line parsed from the latched data. NULL string if end of data is encountered.

This function will assume the latched data is in CF_TEXT format and parse the data for the next text line from the current read position. The data read position will be updated.

dde_offset Win

Set/Move latched data's read position.

long dde_offset (*chid, offset, origin*)

int *chid* DDE channel ID returned by **dde_open()**.

long *offset* Long value of position offset specifying the number of bytes to move the data read position from the origin.

int *origin* Integer value indicating the origin of the offset position:

0 Offset from the beginning of the data.

- 1 Offset from the current read position.
- 2 Offset from the end of the data.
- 1 Special function call to discard the data.

Return -1L, if no data available, else the current read position.

This function is used to set or to read the current read position of the latched data. The initial read position of the data is set to the beginning of the data when it is received and latched in the internal memory buffer. Subsequent reading of this data using **dde_get()** and **dde_getline()** will update this read position automatically.

dde_open Win

Open a DDE channel and link with specific service on specific topic.

int **dde_open** (*service*, *topic*)

char service[] Name of the DDE server or application to link with.

char topic[] Name of the topic to communicate with the server.

Return DDE channel ID in positive integer value if the operation is successful. Otherwise, one of the following error codes is returned:

- 0 Fails to connect with the named service. The specific server may not exist or the specific topic is not accepted by the server.
- 1 Fails to register atom of name, program instance, etc., due to memory shortage.
- 2 DDE channel is not available. All channels are in use.

This function is used to establish a DDE link with a specific DDE server. If the specific DDE service exists and the topic is accepted, it returns an ID number of the established channel of DDE data link. This ID number should be used to identify the DDE channel in subsequent DDE channel data I/O function calls.

Once a DDE channel is opened successfully, it will remain opened until it is closed explicitly via **dde_close()** function call. The termination of a TCL application will not cause any opened DDE channel be closed automatically, even if the channel is opened by the terminating TCL application. So, if a TCL application should fail to close the DDE channel it opened before its termination, it is possible to use out all the available DDE channels and cause the subsequent **dde_open()** call to return -2 error code. However, the **dde_closeall()** may be called to close all the DDE channels and make them available again.

The total number of available DDE channels may vary with version. Currently, at least 8 channels are provided.

Every Windows-based application that supports DDE must have a unique DDE application name, which is not necessarily the name of the executable file for it without the EXE filename extension. If you are not sure what DDE application name is for a specific application, check its documentation.

dde_poke Win

Update a specific data item on the DDE server.

int **dde_poke** (*chid, item, data*)

- int** *chid* DDE channel ID returned by **dde_open()**.
- char** *item[]* Name of the data item to update in CF_TEXT format.
- char** *data[]* String data to update the specific item on the server in CF_TEXT format.
- Return** True, if the operation is successful; False if it fails.

This function is used to update the content of a specific data item on the server using CF_TEXT format. Another function call to update data in other formats is given below:

int **dde_poke** (*chid, item, format, data...*)

- int** *chid* DDE channel ID returned by **dde_open()**.
- char** *item[]* Name of the data item to update in CF_TEXT format.
- int** *format* Data format ID. See **cb_format()** for obtaining the customer data format ID.
- Var** *data* Variable number of arguments (maximum to 15) of accepted data types: POINT, LINE, ARC, CIRCLE, LTYPE, LAYER, and all elementary data types such as char, int, long, double, etc. Note that the ENTITY, ENTDATA and other data type not in the previous list will not be accepted as the valid arguments, since their sizes are varied and structural contents are subjected to change.
- Return** True, if the operation is successful; False if it fails.

dde_request Win

Request a specific item of data from DDE server.

long **dde_request** (*chid, item, format*)

- int** *chid* DDE channel ID returned by **dde_open()**.
- char** *item[]* Name of the data item to request.
- int** *format* Optional data format ID. If this argument is not given, the CF_TEXT format is the default. See also **cb_format()** for obtaining the customer data format ID.
- Return** Length of the data received in the internal data latch buffer if the operation is successful. If it returns zero or negative value, it means error.

This function is used to request a specific data item from the server. It will wait until the data is received or a time-out error is encountered. Use **dde_timeout()** to set the time-out constant.

If the requested data is received successfully, it will be latched in a memory buffer. You may use **dde_get()**, **dde_getline()**, and **dde_offset()** to access this data memory. The initial read counter will be set to 0, the beginning of the data.

The latched data will be discarded whenever a new data is received from the same DDE channel. The newly received data will become the current latched data for reading. You may explicitly discard the latched data by using the **dde_offset** function. See **dde_offset** for further details.

Another way to request data is given below:

int **dde_request** (*chid, item, text, tab*)

- int chid** DDE channel ID returned by **dde_open()**.
- char item[]** Name of the data item to request.
- char text[][]** String array of text buffer to receive the data in CF_TEXT format. The received data will be parsed into lines of text string and stored in the string buffer by order.
- int tab** Optional control flag, TRUE, if the TAB character will be taken as a line delimiter, and FALSE, it will be taken as ordinary data. Some DDE servers, such as EXCEL, use TAB character as the string delimiter in the item data. The default is false, which means to take LF as the line delimiter only.
- Return** Number of text string read into the returned buffer.

This function call is used to request a specific data item from the server in CF_TEXT format and directly returns the received data in the string buffer without latching it in the memory buffer. It is much faster and simpler for reading simple data in CF_TEXT format.

Note that no data will be latched in the internal memory buffer after this function call. The previous latched data will be freed.

dde_state Win

Obtain the status of a specific DDE channel.

int **dde_state** (*chid, item, topic, service*)

- int chid** DDE channel ID returned by **dde_open()**.
- char item[]** Optional string buffer to return the item name of the current received data or active data (warm-link).
- char topic[]** Optional string buffer to return the topic name of the DDE channel.
- char service[]** Optional string buffer to return the server's name of the DDE channel.
- Return** The DDE channel's current status. Effective bit flags are given below:
- Bit 0:** Flag value 0x0001, active data item is in warm link with the DDE server.
 - Bit 1:** Flag value 0x0002, active data item is in hot link with the DDE server.
 - Bit 12:** Flag value 0x1000, server has notified that the data item on server has changed.

Bit 13: Flag value 0x2000, channel has data in latch buffer.

Bit 14: Flag value 0x4000, server is connected.

Bit 15: Flag value 0x8000, channel is opened.

This function is used to obtain status information from the specific server.

dde_timeout Win

Set DDE linkage time-out constant.

long dde_timeout (*timeout*)

long *timeout* Long value specifying the new time-out constant in units of ms.

Return Effective time-out constant in units of ms. The internal default is 3000 ms (3 second).

This function is used to specify a global time-out constant for the DDE transaction. That is, it specifies the longest time to wait for the DDE server to complete the transaction.

dde_warmlink Win

Establish a warm link on specific data item.

int dde_warmlink (*chid, item, format*)

int *chid* DDE channel ID returned by **dde_open()**.

char *item*[] Name of the data item to be notified via warm link.

int *format* Optional data format ID. If this argument is not given, CF_TEXT format is assumed. See **cb_format()** for obtaining the customer data format ID.

Return True, if the operation is successful; False, if it fails.

This function is used to establish a warm link with the server on a specific data item. If it is successful, the server will automatically advise the client without the data whenever the data item has changed on the server side. The TCL application may use **dde_state()** to tell whether the server has notified the data change or not, and use **dde_request()** to obtain the data.

To stop a warm link, use **dde_coldlink()**.

delay

Delay program execution for a specific period of time.

int delay (*msec, brkflag*)

long *msec* Delay time in mini-seconds.

int *brkflag* Optional bit flag control, default to 0:

Bit 0: 1, if the delay can be interrupted by user keyboard input. 0, if the delay can not be interrupted.
Else Reserved, should be 0.

Return 0, if time out, else interrupted keypress value. Note that the interrupted key press value is not removed from the key queue.

This function is used to delay program execution for a specific period of time. The same functionality can also be achieved by the call to command("DELAY msec").

Note: It may be necessary to delay the TCL execution for a short time after winexec() to wait for the shelled program to settle down on initialization, before trying to get connection via the DDE link.

DeviceCaps Win

Obtain specific device capability value.

int DeviceCaps (*which*, *ofst*)

int *which* Integer, specifying which device capability to ask for. It must be 0 for the screen device.

int *ofst* Integer, specifying the capability ID value (offset value to the internal control data).

Return Integer device capability data.

This function call Windows API GetDeviceCaps() directly for the screen device (via GetDC(NULL)), when the argument *which* is 0. Interested device parameters are listed below:

```
#define DRIVERVERSION 0
#define TECHNOLOGY 2
#define HORZSIZE 4
#define VERTSIZE 6
#define HORZRES 8
#define VERTRES 10
#define BITSPIXEL 12
#define PLANES 14
#define NUMBRUSHES 16
#define NUMPENS 18
#define NUMMARKERS 20
#define NUMFONTS 22
#define NUMCOLORS 24
#define PDEVICESIZE 26
#define CURVECAPS 28
#define LINECAPS 30
#define POLYGONALCAPS 32
#define TEXTCAPS 34
#define CLIPCAPS 36
#define RASTERCAPS 38
#define ASPECTX 40
```

```
#define ASPECTY      42
#define ASPECTXY    44
#define LOGPIXELSX   88
#define LOGPIXELSY   90
#define SIZEPALETTE 104
#define NUMRESERVED  106
#define COLORRES     108
```

See Windows SDK for more informations on these parameters.

dgt_btn

Obtain the button status of the tablet digitizer.

int **dgt_btn()**

Return The current button status of the tablet digitizer.

dgt_load

Load specific configuration file to tablet driver.

int **dgt_load(char fname[])**

char fname[] Pathname of the configuration file, which is in the same format as the DDS file.

Return 1, if the operation is success, 0 if it fail.

dgt_off

Turn off tablet active input to TwinCAD.

int **dgt_off()**

Return The current status of the tablet.

If the tablet is off, the movement of the digitizer will not move the mouse pointer or cursor pointer on the screen, nor will it activate any tablet menu. However, the TCLApp can still read its position via **dgt_pos()** and button status via **dgt_btn()** function calls.

dgt_on

Turn on tablet active input to TwinCAD.

int **dgt_on()**

Return The current status of the tablet.

If the tablet is on, the movement of the digitizer will move the mouse pointer or cursor pointer on the screen, according to the tablet mapping configuration. The activation of the digitizer button will also be converted into mouse button if it is within the screen area. The tablet menu to **TwinCAD** will become effective.

dgt_pos

Obtain the position of the tablet digitizer.

POINT dgt_pos()

Return The current digitizer's absolute position on the tablet.

The coordinate values so returned are the un-converted values reported by the tablet device. The physical unit of these values depend on the working resolution of the tablet device, which is unknown to TCLApp. The TCLApp may device a calibration procedure to overcome this.

dgt_save

Save current tablet configuration to file.

int dgt_save(char fname[])

char fname[] Pathname of the configuration file to save.

Return 1, if the operation is success, 0 if it fail.

dgt_state

Obtain the status of the tablet driver.

int dgt_state()

Return The current status of the tablet driver, bit-flag coded:

- Bit 0:** 1, if driver is activated, 0, if not.
- Bit 7:** 1, if driver is suspended, 0, if not.
- Bit 15:** 1, if driver is external, 0, if internal.
- Else:** Reserved, and would be zero.

dim_text

Obtain the default dimension text string under current active dimension variable setting.

STRING dim_text (type, value)

int type Type number of the dimension, as listed below:

- 0** Linear dimension
- 1** Angular dimension
- 2** Diameter dimension
- 3** Radius dimension
- 4** Ordinate dimension

double value Dimension value, which will be converted into appropriate ASCII equivalence in the dimension text. Note that, it is in units of radian if for angular dimensioning.

Return Text string of the dimension text generated under current dimension variable setting.

diskfree

Return the size of free disk space on a specific drive.

long **diskfree** (*drvno*)

or

long **diskfree** (*drvspec*)

int drvno Drive ID number, 0: current, 1:A, 2:B, 3:C, ... etc. Optional, default to 0.

char drvspec Optional drive specification in string format as: "d:...".

Return The size of the free disk space in units of 1024 bytes (KB).

div

Intrinsic function to implement the '/' (division) operation.

Var **div** (*args...*)

Var args Variable number of arguments of applicable data type. The minimum number of arguments are 2, while the maximum number are 15. Type conversions are performed automatically from left to right in sequence.

Return Result of the division, depending on the type of arguments.

This is the intrinsic function to implement the '/' operation in the expression. In other words, the expression: *a/b/c/...* can be written as **DIV(a,b,c,...)**. Note that both upper case and low case of the function names are accepted. The type of operands can be integers, doubles, and POINTs. Type conversions from integer value to double are performed automatically when they are mixed in the operation. See the table below:

Operand1	-	integer	double	POINT
Operand2		-----		
integer		integer	double	POINT
double		double	double	POINT
POINT		POINT	POINT	POINT

Dividing a POINT data by a scalar value produces a scaling effect over the POINT with respect to the origin. Note that POINT data is also taken as a complex number.

Also note that the actual implementation of **DIV(a,b,c,...)** is **DIV(a,MUL(b,c,...))**.

dtr

Intrinsic function to convert angle value from units of degree to units of radians.

double dtr (*angle*)

double *angle* Angle value in units of degree.

Return Equivalent angle value in radians.

e or E

Type Casting Function for Entity Handle

ENTITY e (*ent*)

ENTITY *ent* Trivial argument to be returned.

Return The given argument.

Currently, this Type Casting Function is too trivial to use.

eblock

Get the entity handle of a block by its name.

ENTITY eblock (*name,oper*)

char *name[]* String, name of the block to access.

int *oper* Integer, specifying which type of operation is intended, optional and default to 0. The meaning is given below:

- 0** Match the block with the given name, and then return the block found, otherwise return Null. This is the default.
- >= 1** Match the block with the given name, and then return the next one in the system table, otherwise return the first block in the system table. Note that if there is no block in the system table, or the matched one is already the last one, it returns Null.
- <= -1** Match the block with the given name, and then return the one before it in the system table, otherwise return the last block in the system table. Note that if there is no block in the system table, or the matched one is already the first one, it returns Null.

Return Entity handle pointing to the specific BLOCK entity, from which the TCL application may travel the content of its definition by successively calling to **ent_bnext()**. The BLOCK entity data also contains the full name of the block.

You may use this function to test whether a specific block having been built in the drawing database. Note that the valid name of a block must contain alphanumeric characters, '_' (underscore), '\$' (dollar) and '-' (minus) characters only. The name will be turned into upper case to search for the block.

The block name may be obtained from the field of an existing block instance (**INSERT**). Obtain the entity handle of a block so that the TCL program may travel through the block definition by **ent_bnext()** function call.

This function also accepts an entity handle pointing to an INSERT/REGION entity as its first argument and returns the block control entity directly. Note also that, if a name is given as the first argument and no blocks can be found to match it, the regions will be searched for the match.

An example usage of this function to list out all the blocks in the drawing is given below:

```
e1 = eblock("",1);
if ( ent_ok(e1) )
    printf("\nThe following Blocks are defined: \n");
    while(ent_ok(e1))
        s1 = ent_read(e1).block.name;
        printf("\n %s",s1);
        e1 = eblock(s1,1);
    }
}
else printf("\nNo block in system\n");
```

ecs_org

Setup ECS's origin and X-axis direction for ECS transformation.

int **ecs_org** (*porg,zorg,pxdir,zxdir*)

- POINT porg** X/Y component of the ECS Origin in WCS coordinate.
- double zorg** Z component of the ECS Origin in WCS coordinate.
- POINT pxdir** X/Y component of the ECS X-axis direction in WCS coordinate.
- double pzdir** Z component of the ECS X-axis direction in WCS coordinate.
- Return** 1, if ECS plane is setup, 0, if it is not.

TCL programmers may setup the desired ECS plane by the **set_ecs()** function and setup the origin and X-axis direction on the ECS plane by the use of this **ecs_org()** function. All subsequent creation of 2-D entities will be transformed to the ECS plane with the given origin and orientation.

Note that the creation of entities may also be introduced by such prime commands as COPY, MIRROR, OFFSET, ... etc, which create new entities by duplicating and transforming from existing entities. However, since the transformation takes the X-axis in WCS for alignment to the new ECS, duplication of an entity on an existing ECS will result in complicated coordinate transformation, which is not yet properly handled at the time being. As the current release of **TwinCAD** does not explicitly support 3-D operations, this would not be a problem to the user. Nevertheless, those interested TCL programmers that may utilize these function to create 3-D objects must be aware of this deficiency.

The function **wto_ecs()** and **eto_wcs()** are also affected by this function. The transformation will take this ECS origin and orientation into consideration. However, the coordinate values read from or written to an entity data that lies on an ECS plane, must

be relative to a normalized ECS plane, which takes the WCS origin as its origin and a default X-axis direction determined by the system on certain rules.

With the help of **set_ecs()** and **ecs_org()** function calls, the operator may create objects in 2-D and then use COPY to create those objects on specific ECS plane with the desired alignment of orientation and location. Of course, the TCL programmers may do further more.

To reset the ECS origin and orientation setting to a normalized ECS plane, use **ecs_org(0)**, or **ecs_org(p(0,0),0,p(0,0),0)**.

edrvtype

Obtain the identification number of the loaded extended font driver.

int edrvtype()

Return Integer value of the identification number of the loaded extended font driver, as listed below:

0	None. No extended font driver is active. All text characters in the range of 128 to 255 will be treated as the extended character of IBM PCs character set.
1	Big5-coded Chinese.
2	GB2312-coded Chinese.
3	Korean.
4	TIS-coded Thai.
5	Japanese, JIS coded.
6	Vietnamese, VNI coded.
else	Reserved for other drivers.

Windows Specific

This function is simulated by converting the returned value from the **os_language()**. TwinCAD does not need to load any font driver for screen font display in Windows.

ee_sline

Determine the shortest line between two given objects.

LINE ee_sline (e1,e2,angle,flag)

Var e1 The first object. Valid arguments are POINT, LINE, ARC, CIRCLE, entity handle pointing to POINT, LINE, ARC, CIRCLE, and POLYLINE.

Var e2 The second object, ditto.

double angle Optional measuring direction, units in degree. If this optional angle is not given, the shortest distance will be the shortest distance for all possible angle of direction.

int flag Optional flag, default to 1 if **angle** is given and 0 if **angle** is not given, effective only after version V3.1.054, to specify addition transformation control, as described below:

- Bit 0** 1, if in fix angle direction, 0, if in free angle direction,
- Bit 4** 1, if **set_trns()** is effective to **e1**, 0, if not.
- Bit 5** 1, if **set_trns()** is effective to **e2**, 0, if not.
- Else** Reserved and should be zero.

Return LINE data from a point on **e1** to a point on **e2** where the required shortest distance is located.

If the two objects intersects, the shortest distance will be zero. the returned LINE data will be invalid (from p(0,0) to p(0,0)). Likewise, if an angle direction is specified and there is no possible shortest line to join the two objects in the specified direction, the returned LINE data will also be invalid.

It is possible to have more than one shortest line joining the two given objects (for example, two parallel lines), however, only the first one encountered will be returned (along the direction of the first object).

This is an advanced TCL function. The use of this function is intended for specific applications.

eid

Convert entity handle to integer order number.

long eid (ent)

ENTITY ent Entity handle.

Return Creation order number in drawing database.

This is a hidden function from **CMDLIST**, and is intended for special use.

ent_alloc

Allocate empty entity list of given size.

SLIST ent_alloc (size)

int size Size of entity selection list in units of entity number.

Return Entity selection list of specified size with empty content.

Note that an entity selection list is basically an array of entity ID, and is dynamically allocated from the TCL data stack. A variable of such **SLIST** type is a pointer by its natural property, pointing to the allocated array. To reference each element from the selection list, the program must use the array subscript method, since no pointer references are supported yet in current TCL version.

Also note that an entity array is not equivalent to an entity selection list, though the latter actually contains an array of entity ID. This is because the structure of an ENTITY type data contains other information than the entity ID, such as the pick point.

Another limitation that needs to be aware of on the use of such dynamically allocated data aggregates is their lifetime of validities, which last for the same interval as the local variables do in the same function body where the allocations are made. They will be de-allocated from the data stack automatically at the time the function returns.

ent_area

Calculate bounded area by entity or entities.

double ent_area (*argument*)

Var *argument* Variable number of arguments of predefined type. Different number of arguments of different type will specify different area calculation. See the paragraphs below.

Return Area bounded by arguments. Note that the returned area value may be negative. If it is positive, the direction of area boundaries is counter-clockwise. If it is negative, then the direction is clockwise.

The valid arguments for the area calculation are described below:

- ps,pm,pe** Calculate the area bounded by 3 points, from *ps*, through *pm* to *pe*. The three arguments must be of type POINT. The returned area will be positive, if the arc starting from *ps* through *pm* to *pe* goes counter-clockwise. It is zero, if the three points lie in the same line. It is negative, if the arc passing through them goes clockwise.
- pbase,ln** Calculate the area bounded by a line with respect to a base point. The first argument must be of POINT type, while the second one, of LINE type or an entity handle pointing to a line. Note that only the 2D portion of the line is used for the calculation. Again, the sign of the returned area shows the line pointing direction with respect to the base point, as in the case of given three points.
- pbase,arc** Calculate the area bounded by an arc with respect to a base point. The first argument must be of POINT type, while the second one, of ARC type or an entity handle pointing to an arc. The area is bounded by the arc segment, and the lines from the base point to its two end points. The returned area has sign.
- ccl** Calculate the area bounded by a circle. The argument must be of type CIRCLE or an entity handle that points to a circle. The returned value is always positive.
- polyline** Calculate the area bounded by a polyline. The argument must be an entity handle that points to a polyline. If the polyline was not twisted, then the sign of the returned area shows the direction of the polyline. Positive area indicates the polyline going counter-clockwise, while negative area, clockwise.

ent_bnext

Get next entity belonging to the same level of BLOCK.

ENTITY ent_bnext (*ent*)

ENTITY *ent* Entity handle pointing to an entity that is either a BLOCK or belonging to a BLOCK.

Return If the given entity is a BLOCK, then the first entity belonging to it is returned. Otherwise, the next entity that is also belonging to the BLOCK is returned. If the given entity is not valid, or it is the last entity of the BLOCK, the function returns null.

ent_break

Break a given entity at specific point.

ENTITY **ent_break** (*ent,pnt*)

ENTITY *ent* Entity handle pointing to an entity that is to break into parts. Valid entities are LINE, ARC, CIRCLE, POLYLINE.

POINT *pnt* Break point, not necessary exactly on the given entity.

Return Null, if the operation fails due to the given break point can not be used to separate the entity; otherwise, the entity handle to the newly created entity by separating the given entity, or the original entity if the entity was only modified (from close to open).

If the given point does not exactly lie on the given entity, the nearest point on the entity to the given point is used as the point of separation. If this point of separation being located is outside of or at the end point of the given entity, the operation fails.

If the given entity is a segment of a polyline, then the separation point will be on this segment, determined by the given point. If the nearest point to the given point from this segment is outside of the segment, the nearest end point of it will be used to break the polyline.

ent_copy

Duplicate selected entities with newly setup properties.

SLIST **ent_copy** (*ent,...*)

Var *ent* Entity handle or selection list of entity to be copied.

Return Selection list of the newly created entities.

This function accepts variable number of arguments from 1 to 15. Each argument should be an entity handle (must be a valid one) or a selection list. It will duplicate these given entities in such a way that the newly duplicated entities will:

. Assume new properties of Layer, Color, and Linetype from the current setup of properties for new entity creation if the @USE_EMODE is ON; otherwise, properties are also copied. (If the TCL version number is less than 301, the @USE_EMODE is always assumed ON).

. Assume new Elevation and Thickness, if applicable, from the current setup values for new entity creation.

. Assume new Entity Plane (ECS coordinates), if the ECS is enabled (by **set_ecs()** and **ecs_org()**).

. Be transformed by current geometry transformation setup specified by **set_trns()**.

ent_count

Count the number of entities in a selection list.

int ent_count (entlist)

SLIST entlist Entity selection list. After V3.1.058+, if this argument is missing, all the top-level drawing entities in the drawing database are assumed.

Return The number of entities in the selection list. If the given argument is not valid or is not yet properly initialized, or has become invalid, the returning number will be zero.

After V3.1.058+, if the argument is missing, it will return the total number of the top-level drawing entities in the drawing database.

int ent_count (ent)

ENTITY ent Entity handle of a POLYLINE or BLOCK entity.

Return The number of component entities. Note that this is not a new function features in current release; it was just not documented in previous releases.

Function features supported only after V3.1.058+:

int ent_count (eMain, eStop)

ENTITY eMain Entity handle of a POLYLINE or BLOCK entity.

ENTITY eStop Optional handle of component entity of *eMain* to stop the counting. If this argument is missing or it is not a valid component of the *eMain*, **ent_count()** counts to the end.

Return The number of component entities counted as required.

long ent_count ()

Return The number of top level entities in the drawing. This function call will always return 0 in previous releases that do not support it.

ent_erase

Erase specific entities

int ent_erase (ent,...)

ENTITY ent One or more of arguments of ENTITY type or SLIST (Entity Selection list) type to be erased from the drawing database.

Return Number of entities being erased.

Note that once an entity is erased, the entity handle associated with it will become invalid (null). Also note that you can not erase a block entity by any means.

ent_init

Initialize an ENTDATA with given parameters.

ENTDATA **ent_init** (*type, parms,...*)

int type Integer value specifying the type of entity to initialize. See **ent_type()** for a list of supported entity type value.

VAR parms Optional number of variable arguments used to initialize the entity data. The format details depend on the type of entity to initialize. See later sections for details.

Return ENTDATA initialized with the current layer, color and linetype. All data fields will be filled with zero, except those explicitly initialized with additional arguments.

This function is used to initialize an ENTDATA for complex drawing entity creation or generation. The TCL application may need to further fill out the data fields of the ENTDATA to have a well defined ENTDATA. Without using this function, TCL application can obtain an initialized ENTDATA only via the **ent_read()** function call, which requires that the drawing entity has been created in the drawing database.

The following are a list of **ent_init()** function call prototype with initialization parameters:

**** Linear Dimension:** ALDIM/HDIM/VDIM, result affected by the current **PUCS** setup.

ENTDATA **ent_init** (*9,0,ps,pe,pt,ob,rot,opt*)

ENTDATA **ent_init** (*9,0x10,ps,pe,pt,ob,rot,opt*)

ENTDATA **ent_init** (*9,0x20,ps,pe,pt,ob,rot,opt*)

POINT ps First dimension point.

POINT pe Second dimension point.

POINT pt Text position (dragged position).

double ob Optional oblique angle if applicable.

double rot Optional rotated angle if applicable (ALDIM).

int opt Optional flag to override DIMLOPT.

**** Angular Dimension:** Result unaffected by the current **PUCS** setup.

ENTDATA **ent_init** (*9,1,pc,ps,pe,pt,opt*)

ENTDATA **ent_init** (*9,1,cc,ps,pe,pt,opt*)

ENTDATA ent_init (9,0x14,po,pm,pt,pl)

POINT po Ordinate origin.

POINT pm Measuring point.

POINT pt Tail position (text position).

POINT pl Optional lead line position.

**** Leader:** Pure leader (5) with/without text, Spline leader (0x15) with/without text, result unaffected by the current **PUCS** setup.

ENTDATA ent_init (9,5,p1,p2,...)

ENTDATA ent_init (9,5,str,pt,angle,p1,p2,...)

ENTDATA ent_init (9,0x15,p1,p2,...)

ENTDATA ent_init (9,0x15,str,pt,angle,p1,p2,...)

char str[] Short Text Anotation, maximum 40 chars.

POINT pt Text insertion position.

double angle Angle of text insertion.

POINT p1, p2 First point (leader head) and second point, etc. The maximum vertex point number for leader with text anotation is 5, and without text anotation is 8.

**** Center Mark:** Result affected by the current **PUCS** setup.

ENTDATA ent_init (9,6,cc,len)

CIRCLE cc Circle to mark.

double len Length specification of the mark.

**** Insertion:** Block instance.

ENTDATA ent_init (8,name,pbase,pscale,rot)

char name[] Name of the existing block to insert.

POINT pbase Insertion base point.

POINT pscale Scale factors along each X and Y axis.

double rot Rotation angle.

ent_intsc

Calculate intersection point of two entities.

int **ent_intsc** (*emt1,emt2,pint,type*)

VAR emt1, emt2 LINE, ARC, CIRCLE, ENTITY type of argument of elementary geometry entities, include polylines.

POINT pint[] Array of at least 2 elements to hold the possible returned points of intersection.

int type Specifying the type of point returned. If it is omitted or 0, then only point on both elements be regarded as point of intersection. Otherwise, all mathematical points of intersection are returned as the given geometries are extended. However, if either one of the argument is a polyline, the intersection points returned must be on the polyline.

Return The number of intersection points is returned. Maximum 128 points can be returned by **ent_intsc()**.

ent_last

Get last entity from the drawing database.

ENTITY **ent_last** (*ent*)

ENTITY ent Optional entity handle, if present, the entity which comes before it at the main level of the drawing database is returned.

Return If no argument is not present, the last created entity at the main level of the drawing database is returned. Otherwise, the entity which comes before it at the main level of the drawing database is returned. It returns null if the entity is already the first entity in the database.

Note that this function will never return an entity of BLOCK, a block element, a polyline element, or any element that is part of a composite entity. It always return entity of the main level, regardless of the level of the given argument.

ent_len

Calculate the length of given geometry entity.

double **ent_len** (*ent*)

ENTITY ent Entity to calculate the length. Valid arguments are geometry entity like POINT, LINE, ARC, CIRCLE, ELLIPSE and entity handle pointing to LINE, ARC, CIRCLE, ELLIPSE and POLYLINE. If the argument is a POINT, it return the distance from the point to the origin (0,0).

Return Length of the entity.

This function has been provided in **v()** or **V()** function calls, but most of the programmers may not have noticed this fact.

ent_main

Obtain the upper level entity of a given entity.

ENTITY **ent_main** (*ent*)

ENTITY *ent* Entity handle points to a valid entity that may be part of an composite entity.

Return Entity handle points to the upper level entity of the given entity. Return null if the given entity is already at the main level.

This function is used to locate a polyline from its line segments or a tagged entity by a tag entity.

ent_next

Get next entity from the drawing database.

ENTITY **ent_next** (*ent*)

ENTITY *ent* Optional entity handle, if present, the entity which comes after it at the main level of the drawing database is returned.

Return If no argument is not present, the first entity at the main level of the drawing database is returned. Otherwise, the entity which comes after it at the main level of the drawing database is returned. It returns null if the entity is already the last entity in the database.

Note that this function will never return an entity of BLOCK, a block element, a polyline element, or any element that is part of a composite entity. It always return entity of the main level, regardless of the level of the given argument.

ent_null

Returning a null pointer of a entity handle, used to make an entity handle become invalid.

ENTITY **ent_null**()

Return Null pointer of an entity handle.

Note that you can't assign a zero to an entity handle to make it point to nothing, because it is a structure in **TCL**.

ent_ofst

Generate the offseted geometry.

Var **ent_ofst** (*ent,dir,dist*)

or

Var **ent_ofst** (*ent,dir,pnt*)

Var ent	Geometry entity used as a reference to calculate the offsetted geometry. Valid arguments are LINE, ARC and CIRCLE and entity handle pointing to LINE, ARC, CIRCLE and POLYLINE. Note that the direction of a circle is always assumed CCW, so to offset right (+) will enlarge the circle.
int dir	Offset direction. If this value is positive, the offset path will be generated to the right of the given entity, and the third argument must be the offset value (double). If this value is negative, the offset path will be generated to the left of the given entity, and the third argument must be the offset value (double). If this value is zero, the third argument must be a POINT, and the offset path will be generated to pass through the given POINT.
double dist	Distance to offset, if dir is not zero.
POINT pnt	Point coordinate where the offset path must pass through, if dir is zero. The rules of this Through Point offsetting are the same as those for OFFSET command.
Return	Offsetted geometry data of the same type as the first argument

If the given entity is an entity handle, new entity will be created and the new entity handle will be returned. If the returned entity handle is NULL, then no offset path is found (especially in offsetting a polyline).

The rules used in generating the offset path are the same as the **OFFSET** command. See also **OFFSET** command.

ent_ok

Test if the given object is a valid entity.

int ent_ok (ent)

Var ent	An entity handle, or a real entity structure to test for validity. See explanation in later paragraphs.
Return	True if the given entity is valid, false if it is not.

Virtually everything is accepted by this function as an argument for the validity test. The function returns true or false to reveal whether or not the argument is valid of its type.

The scalar data (integer, long, double) are always valid. Points are always valid too. A circle is valid if and only if its radius value is positive and not zero. A line is valid if both of its end points are distinct. An arc is tested by its spanning angle as well as its radius. An arc of zero spanning angle is invalid.

As for entity handles or entity data, they are tested to see if they were properly initialized. An entity data must be initialized to become valid by the function **ent_read()** with a valid entity handle. A valid entity handle must not be null, must have been initialized to point to the real entity in the drawing database by means of library function calls, such as **gete()**, or assignment from the entity selection list. Note that an entity handle becomes invalid if the entity it points to has been erased.

String is also accepted by this function for the test. The length of the string is returned as a test result, which means that the string is valid if it is not a null string.

ent_plast

Get the last segment of a polyline or polyline segment.

ENTITY ent_plast (ent)

ENTITY ent Entity handle pointing to a POLYLINE, or an entity that is part of a POLYLINE.

Return If the argument points to a POLYLINE, then the last segment of the POLYLINE is returned. If it already points to a segment belonging to a POLYLINE, then the segment comes before it is returned. It return null if the argument is not a POLYLINE or polyline segment, or it is already the first segment of the POLYLINE and the POLYLINE is open.

Note that if the POLYLINE is close, this function will not return null when it travels over the starting segment, but the last segment of the POLYLINE is returned. The application program must check for the ending conditions by explicit checking against the entity handle.

ent_pnext

Get the next segment of a polyline or polyline segment.

ENTITY ent_pnext (ent)

ENTITY ent Entity handle pointing to a POLYLINE, or an entity that is part of a POLYLINE.

Return If the argument points to a POLYLINE, then the first segment of the POLYLINE is returned. If it already points to a segment belonging to a POLYLINE, then the segment comes after it is returned. It return null if the argument is not a POLYLINE or polyline segment, or it is already the last segment of the POLYLINE and the POLYLINE is open.

Note that if the POLYLINE is close, this function will not return null when it travels over the last segment, but the first segment of the POLYLINE is returned. The application program must check for the ending conditions by explicit checking against the entity handle.

ent_pseg

Find the segment of a polyline that contains a specific point.

ENTITY ent_pseg (ePoly, pos)

ENTITY ePoly Entity handle pointing to a valid POLYLINE.

POINT pos Point on this polyline to query.

Return Entity handle of the segment that contains the given point. Null handle is returned if the given point is not on any segment.

To find the nearest point on a polyline to a given point, use "L(ee_sline(pos,ePoly)).pe".

This function is supported only after version V3.1.058+. In fact, an equivalent implementation of **ent_pseg()** in TCL is given below:

```
ENTITY ent_pseg(ENTITY ep, POINT pos) @PRINTOUT =
    ENTITY    eFirst, eCur;
    if ( ent_type(ep) != T_POLY ) return ent_null();
    eCur = eFirst = ent_pnext(ep);
    while(ent_ok(eCur))
        if ( in_span(eCur,pos) )
            if ( pe_dist(pos,eCur) == 0. )
                return eCur;
        }
        eCur = ent_pnext(eCur);
        if ( eid(eCur) == eid(eFirst) ) break;
    }
    return ent_null();
}
```

ent_read

Read entity data from drawing database.

ENTDATA ent_read (*entid*)

ENTITY *entid* Entity handle, obtained by other function calls.

Return Entity data from the drawing database.

Only after the object data being read from the drawing database, can the individual field of data of an entity be address for read access.

ent_rvs

Reverse an entity's direction.

Var ent_rvs (*ent*)

Var *ent* Entity such as LINE, ARC, ELLIPSE, or entity handle pointing to Line, Arc, Elliptic arc and Polyline.

Return It return the same data type as the argument. If the given argument is an entity handle, the direction of the entity in the database will be reversed and the same entity handle is returned. Note that only LINE, ARC, LINE3D and POLYLINE can be reversed and a single polyline segment can not be reversed individually.

ent_state

Modify or read the status of an entity.

int ent_state (ent, cmd, flag)

ENTITY ent Entity handle that points to the entity to read or to modify its current status.

int cmd Integer specifying how to modify the current status of the entity.

0 Clear the status specified by *flag*.

1 Set the status specified by *flag*.

2 Toggle the status specified by *flag*.

else Nothing.

int flag Integer status flag value to modify. Effective and useful flag values are not available currently.

Return: The last status value of the entity before modification.

ent_tnext

Get next Tag entity that follows.

ENTITY ent_tnext (ent)

ENTITY ent Optional Entity handle that points to an entity that has tag attributes followed. If this argument is not given, the first attribute tag of drawing level will be returned, if it does exist.

Return The next TAG entity that follows. Return null if there is no TAG entity followed. Note that, successive traveling of a tag list is also done by successively calling this function with the returned tag entity handle, until a Null handle is returned.

ent_trns

Entity Geometry Transformation.

Var ent_trns (ent)

Var ent Geometry Entity such as POINT, LINE, ARC, CIRCLE, ELLIPSE, ENTDATA, Entity handle or Selection set.

Return If the argument is a geometry data (such as POINT, LINE, ARC, CIRCLE, ELLIPSE or ENTDATA), it returns the transformed geometry data of the same type. If the argument is an entity handle, the entity in the drawing database will be transformed, and the same entity handle is returned. If the argument is a selection set, the entities in the selection set will be transformed, and the number of entities being transformed is returned.

This is a very useful function. Just use the **set_trns()** function to setup the required transformation matrix, then you may transform anything you like. An example is given below:

```
set_trns(....);  
p1 = ent_trns(p1);
```

```
l1 = ent_trns(l1);  
c2 = ent_trns(c1);  
ent_trns(e1);
```

For a large number of entities pointed by entity handles to be processed by **ent_trns()**, it is recommended to create a selection list to contain them first, and then pass the selection list to **ent_trns()** for faster performance and less overhead in UNDO buffer.

See also **set_trns()** function for further details.

Note: **ent_trns()** will return ELLIPSE data type for ARC/CIRCLE data if the both X/Y scale factors are different. However, entity handle pointing to CIRCLE can not be properly transformed to ELLIPSE in current release of **ent_trns()**, since the transformation will require to erase the old entity handle and create new entity of ellipse.

ent_type

Identify the type of entity represented by the argument.

int ent_type (ent)

Var ent ENTITY or ENTDATA type, object to be tested.

Return Integer value of entity type number (#include tclcad.h). If the argument is not valid, or the entity handle or data is not yet initialized, it return **-1**.

Defined in the include file, **tclcad.h**, are the following macros for currently supported entity type in TCL:

```
#define T_POLY    0    /* Is a polyline (header) */  
#define T_POINT  1    /* Is a point */  
#define T_LINE   2    /* Is a 2-D line */  
#define T_ARC    3    /* Is an Arc */  
#define T_CIRCLE 4    /* Is a circle */  
#define T_LINE3D 5    /* Is a 3-D line */  
#define T_TEXT   6    /* Is a text insert */  
#define T_BLOCK  7    /* Is a block definition */  
#define T_INSERT 8    /* Is a block instance */  
#define T_DIM    9    /* Is a dimension */  
#define T_REGION 10   /* Is a region image */  
#define T_TAG    11   /* Is a tag attribute */  
#define T_SYMBOL 12   /* Is a symbol reference */  
#define T_3DFACE 13   /* Is a symbol reference */  
#define T_ELLIPSE 19  /* Is an Ellipse or Elliptic Arc */
```

ent_write

Write modified entity data back to drawing database.

ENTITY ent_write (ent,obj)

- ENTITY *ent*** Entity handle points to entity to update.
- ENTDATA *obj*** Entity data to update.
- Return** The entity handle after modification, most of the case, will be the same as the given entity.

Note that you must initialize the ENTDATA by **ent_read()**, and then modify the data fields of interests, and then write it back to where it belongs. It is not allowed for different type of entity data be written back. To change a given entity from one type to another, simply erase the entity first and then create the new entity of the required type.

eto_wcs

Transform a 3D point from ECS to WCS.

int **eto_wcs** (*Pe,Ze,pw,zw*)

- POINT *Pe*** X/Y component of the 3D point in ECS.
- double *Ze*** Z component of the 3D point in ECS.
- POINT **pw*** Pointer to a point where the X/Y component in WCS is returned after transformation.
- double **zw*** Pointer to double where the Z-component in WCS is returned after transformation.
- Return** 1, if the transformation is OK, 0, if it fails due to ECS is ineffective.

The ECS must be setup by **set_ecs()** before calling this functions. See also **ecs_org()**, **wto_ecs()** functions for related informations.

eval

Evaluate a string.

Var **eval** (*str*)

- char *str[]*** String to be parsed and evaluated.
- Return** Depending on the result of evaluation, it may return anything.

exbuf_io

Save/Retrieve the exchange informations to/from the system information exchange buffer.

int **exbuf_io** (*cmd,buf,size*)

- int *cmd*** 0 to save information to buffer, 1 to retrieve it from the buffer. All else values are reserved and ignored at present.

char buf[] Data buffer to hold the information to save to or retrieve from the system buffer. Require pointer expression if to retrieve informations.

int size Optional integer, specifying the size of information to I/O in units of bytes. The default is 128.

Return Number of bytes saved to or retrieved from the buffer.

DOS Specific

The system information exchange buffer is a memory area allocated by the **TCAM/Graphic Runtime** to provide an information exchange channel between applications that call the Graphic Runtime. The maximum size of this memory area shall depend on the version of the Graphic Runtime in use. Currently, the size limit is 128 bytes.

As the **TCAM Application Development System (TADS) for DOS** is not released, the use of this function is quite limited. However, the example program fragment given below will help to understand the idea of this function:

```

exbuf_io(0,"Default Text"); // Set initial default text
len=exec("BIG5PHIN.EXE"); // Call for user input of text
exbuf_io(1,&s1);           // Read the user input text
...
    
```

Note that BIG5PHIN.EXE is an **TADS** application that handles the BIG5 Chinese Text input. It will obtain the initial default text from the buffer, and let the user to edit it or replace it with new text, and then put back the text string to the buffer again before it leaves the system. The exit code of this application indicates the length of the text string. This is also how the system calls the text input handler to input text in **TEXT** command.

Windows Specific

The 128 bytes information exchange buffer is simulated in a compatible way so that these TCL applications using this function to exchange informations will work. However, these DOS utilities or text input handlers written with TADS can not be shelled out to exchange informations with TCL applications in Windows environment, since they require the TCAM/Graphic Runtime to be loaded in the system.

exec

Fork out a DOS/Windows application program.

int **exec** (*pgm, parm, outfile, infile*)

char pgm[] Full pathname of the program with explicit file extension.

char parm[] Parameter string as the command line to the program.

char outfile[] Optional output filename for **stdout** output redirection.

char infile[] Optional input filename for **stdin** input redirection.

Return Integer value, if negative, representing error code returned by the system as it failed to fork the program out; otherwise, if 0 or positive, representing the exit code of the successful forked out program.

DOS Specific

This function is intended for running programs related to the TCL applications. For DOS and other applications, use **command("DOS ...")** function. See the notes below:

- The system will not exit the graphic mode when this function is called. However, it will restore the screen back if it had been altered or its mode had been changed.
- If the optional *outfile* is given and is valid, it will be created and the **stdout** of the shelled program will be redirected to it. Otherwise, the **stdout** for the shelled program will be redirected to the **NUL:** device.
- If the optional *infile* is given and is valid and present, it will be opened and the **stdin** of the shelled program will be redirected to read data from it. Otherwise, the **stdin** of the shelled program will remain the same as the system's **stdin**.
- Never use the same name for the both *infile* and *outfile*! Otherwise, the shelled program may hang.
- Since the program is forked out directly without a copy of COMMAND.COM, the I/O redirection ("*>*file or *<*file") specified in the command line may fail, unless the forked program can handle the redirection by itself.
- The system will release most of the memory starting from the allocated buffer after the **TCAM Graphic Runtime** (See the memory map generated by **MEM/M** command). This will be about the total system available memory minus 390KB, which is still retained by the system. For a 590KB working system, this will release about 200KB for the forked program to run in real mode. Actual size of memory that can be released for the program depends on the version of the system and on the system memory configuration. Should the memory size be too small for your application program, use **command("DOS ...")** function. The DOS command will swap the system out of the memory, and release almost everything to a dos shell.
- This function will now automatically search for the program specified in the first argument through the paths specified by the TCADPATH environment variable. However, the TCL application may use **fsearch()** to search for the program through the paths specified by other system environment variable, before calling this function.
- The forked program must not be a TSR application. Or, the system may fail to continue properly.
- Memory check against allocation error will be done after the exit of the forked program. Should an error be found, the function will try to restore and fix the DOS MCB chain, and a message will be given to notice this.

Windows Specific

The **exec** will call **Windows API WinExec()** to fork out the specified program and wait for its termination if it is shelled successfully. A few notes are given below:

- If *parm[]* is given, it is concatenated to the command line for the **WinExec()**.
- If the optional *outfile* is given, it will be concatenated to the command line using "*>*outfile" format. It works only when the shelled program supports the file redirection.
- If the optional *infile* is given, it will be concatenated to the command line using "*<*infile" format. It works only when the shelled program supports the file redirection.

- **exec()** waits for the shelled program to terminate by monitoring the returned Instance Handle from **WinExec()**. It would fail on NT machine in current version, since TwinCAD runs in WOW on NT machine.

For more details information see **winexec()**.

exit

Exit the **TCL** program execution immediately.

void **exit(ret)**

int ret Optional exit code, default 0. The **TCL** application may retrieve the exit code of the latest executed **TCL** program from the system variable @EXITCODE.

The execution of this function call will force the **TCL** to close all the opened resources, and de-allocate all the **TCL** local variable space, and then return control back to **TwinCAD** command line.

exp

Intrinsic function to calculate the exponential function, i.e., raise the base of the natural logarithms to the power of the argument.

double **exp (value)**

double value Power value to raise to the base of natural logarithms.

Return The calculated result in double.

or

POINT **exp (z)**

POINT z Power value in complex number to raise to the base of natural logarithms.

Return The calculated result in complex number (**POINT**).

fclose

Close an open stream (file).

int **fclose (fid)**

FILE fid Stream ID (integer in actual implementation).

Return 1, if the operation succeeds, 0 if the fid is invalid.

fcloseall

Close all open streams (files).

int **fcloseall()**

Return The number of streams being closed.

feof

Test if End-Of-File has been detected while reading from the input stream.

int **feof** (*fid*)

FILE *fid* Stream ID (integer in actual implementation).

Return 1, if EOF has been encountered. 0 if not.

fexist

Test whether a specific file/directory exist or not.

int **fexist** (*fname,smode*)

char *fname*[] Path name of the file/directory to test. The name must be a valid DOS filename or directory name, and may include wildcards (if so, the attribute of first match will be returned).

int *smode* Optional file search mode, default to 0 (search only normal files). See also return code for the search mode value.

Return 0, if the specific file/directory not found or error; otherwise, the attribute code of the file/directory is returned:

- Bit 0** File is ReadOnly
- Bit 1** File or directory is Hidden
- Bit 2** File or directory is a System File or Directory
- Bit 3** Filename is a volume label of media in the specified drive.
- Bit 4** Filename identifies a directory, not a file.
- Bit 8** Always 1 to make the return code not zero.
- Else** Reserved, should be zero.

If the return code is 0x100, then the file is a normal file.

If the file search mode is not specified, then only normal files are searched (default to 0). However, if any combination of Hidden (0x02), System (0x04) and Directory (0x10) is specified in the search mode, then the normal files will also be searched for the match.

Example: To identify the existence of a directory

```
if ( !(fexist("D:\\TCAD\\SUPPORT",0x10)&0x10) )
    printf("\nDirectory does not exist!");
```

fget

Read binary object data from input stream.

int **fget** (*fid,args,...*)

- FILE *fid*** Stream ID (integer in actual implementation).
- Var* *args*** Variable number of pointer arguments (maximum to 15) of accepted data types: POINT, LINE, ARC, CIRCLE, LTYPE, LAYER, and all elementary data types such as char, int, long, double, etc. Note that the ENTITY, ENTDATA and other data type not in the previous list will not be accepted as the arguments, since their sizes are varied and structural contents are subjected to change.
- Return** -1, if EOF is encountered, else the number of bytes read in.

For portability consideration, read a single argument for each call of this function, such that the porting can be done by using macro definition.

If an argument is passed explicitly by pointer (&var), then only a single element of the data type is read in. If an argument is an array, and is given solely by its name (a pointer to the array is passed), then the whole array is read in.

fgets

Read a string from the input stream.

int **fgets (*str,n,fid*)**

char *str*[] Character string to return the string read.

int *n* Maximum number of characters to read.

FILE *fid* Stream ID (integer in actual implementation)

```
char    items[1][28];  
long    lsize; // size of file  
long    ldate; // date/time of file  
get_dirlist(takepath(fname), takename(fname), items);  
memmove(&lsize, &items[0][14], 4);  
memmove(&ldate, &items[0][18], 4);
```

FindWindow Win

Locate a specific window by class name and by caption text.

int FindWindow (*class, name*)

- char *class*[]** Registered class name of the window to find. Null string to match any class of window.
- char *name*[]** Optional, caption text of window to match. If it is not given or null string, then no matching on window caption text.
- Return** Null, if there is no window found to match; otherwise, the window handle of the match.

This function directly call Windows API FindWindow() to locate a specific window. See Windows SDK for more informations.

fit_circle

Calculate the best fitting circle from a set of given points.

CIRCLE fit_circle (*pnt, n*)

- POINT *pnt*[]** Array of points to fit.
- int *n*** Number of points to fit, must be greater than 2.
- Return** The fitting result as a CIRCLE data.

This function applies the least square method to find the best fitting circle among a set of given points.

fit_line

Calculate the best fitting line from a set of given points.

LINE fit_line (*pnt, n*)

- POINT *pnt*[]** Array of points to fit.
- int *n*** Number of points to fit, must be greater than 1.
- Return** The fitting result as a LINE data.

This function applies the least square method to find the best fitting line among a set of given points.

fix

Intrinsic function to round the argument to an integer by truncation (toward zero).

double **fix** (*value*)

double value Double value to round.

Return Double value that is the integer part of the argument.

floor

Intrinsic function to round the argument down to an integer (toward negative infinity).

double **floor** (*value*)

double value Double value to round.

Return Double value that is the largest number not greater than the argument, and that is also an exact mathematical integer. If the value of the argument is already a mathematical integer, then the result equals to the argument.

fopen

Open a specified file for stream I/O.

FILE **fopen** (*filename,mode*)

char filename[] Path name of the file to open.

char mode[] File access mode specifications. Valid modes are:

- "r" Open an existing file for reading.
- "w" Create a new file, or truncate an existing one, for writing.
- "a" Create a new file, or append to an existing one, for writing.
- "r+" Open an existing file for update (both reading and writing), starting at the beginning of the file.
- "w+" Create a new file, or truncate an existing one, for update (both reading and writing).
- "a+" Create a new file, or append to an existing one, for update (both reading and writing).

Return None-zero Stream ID number if the operation succeeds. Zero if it fails to open the file.

fprintf

Formats and prints a series of output characters to the specific output stream.

int **fprintf** (*fid,format[,argument]...*)

FILE *fid* Stream ID (integer in actual implementation).

char *format[]* Format-control string.

Var *argument* Optional arguments of any type as specified in the format-control string.

Return EOF (-1) if the output operation fails, 0 if success.

See **printf()** for a discussion of the formatting operation.

fput

Write binary object data to the output stream.

int **fput** (*fid, args, ...*)

FILE *fid* Stream ID (integer in actual implementation).

Var *args* Variable number of arguments (maximum to 15) of accepted data types: POINT, LINE, ARC, CIRCLE, LTYPE, LAYER, and all elementary data types such as char, int, long, double, etc. Note that the ENTITY, ENTDATA and other data type not in the previous list will not be accepted as the arguments, since their sizes are varied and structural contents are subjected to change.

Return -1, if error found, else the number of bytes written out.

For portability consideration, output a single argument for each call of this function, such that the porting can be done by using macro definition.

If an argument is passed explicitly by pointer (&var), then only a single element of the data type is written out. It is not necessary to use the pointer expression to pass the argument, in the sense of a language interpreter as TCL.

If an argument is an array, and is given solely by its name (a pointer to the array is passed), then the whole array is written out.

fputs

Copies (writes) a string to the output stream.

int **fputs** (*str, fid*)

char *str[]* Character string to be output to the stream. The string must be null terminated.

FILE *fid* Stream ID (integer in actual implementation).

Return EOF (-1) if the output fails, else the number of characters being output.

This function simply copy the string to the output without any interpretation to its content. The "\n" will be sent as a single byte of newline character.

frac

Intrinsic function to obtain the fractional part of a double argument.

double frac (*value*)

double value Double value.

Return Double value that is the fractional part of the argument. Note that the sign is reserved.

fsearch

Searches for the specific file from the paths specified by a system environment variable.

STRING fsearch (*filename, varname*)

char filename[] Name of the target file with optional file path.

char varname[] Optional system variable name (must be in upper case). If this argument is not given, it performs the same search for a file as the TwinCAD does.

Return Null string if the file can not be found, else the file of the specific name found from the path.

The searching starts from the path taken directly from the filename specification if applicable, and then the current drive and directory, and then the paths specified in the given system variable. Once the file is located, it returns the file name with full path name immediately. This function can be used to test whether a given file is present or not by supplying a null system variable.

fseek

Set file position indicator of a stream to a new location.

long fseek (*fid, offset, origin*)

FILE fid Stream ID (integer in actual implementation).

long offset Long value of position offset specifying the number of bytes to move from the origin.

int origin Integer value of 0, 1, or 2, indicating the offset origin at the beginning of file, current file position, and the end of file, respectively.

Return Long value of the resulting file position counter.

To obtain the end of the file, use **fseek(fid,0L,2)** . To obtain the current file position counter, use **fseek(fid,0L,1)** . To move to the beginning of the file, use **fseek(fid,0L,0)** .

fstrloct

Locate a specific string from a file given by handle.

int fstrloct (*fid, str*)

FILE fid Stream ID (integer in actual implementation).

char *str*[] String value to be search from the file.

Return 0, if the string is not found and the file position is not changed.
 1, if the string is found. The file position will be set to the start of the first
 matched string.

This function will search from current file position forward till the first occurrence of the string is encountered, or till the end of file is encountered. If the search fails, the file position will not move.

Searching a file for a specific data in an efficient way require careful design of the file structure in binary format. However, for non-serious TCL application programmers, this function should help a lot in linear searching over an ASCII text file for a specific key string before reading a specific data that follows.

This function directly accesses the file reading buffer, which is 2048 bytes in size, and is optimized in assembly codes for speed. If you still suffer the slowness in accessing data this way, you should consider re-organizing your data file structure in a different format! The slowness does not come from the way how your program is interpreted, but how it was written.

ftell

Obtain the current file position counter of a specific stream.

long **ftell** (*fid*)

FILE *fid* Stream ID (integer in actual implementation).

Return Long value of the file position counter of the specific file stream.

This function can be implemented by **fseek(*fid*,0L,1)** function call.

fullname

Combine path and filename into full path name of file

STRING **fullname** (*fname,path*)

char *fname*[] File name.

char *path*[] Path name, Optional.

Return Combined fullpath name.

If the argument *path* is omitted, or is null, then path of current directory is used.

gauss

Solve the linear set of equations of order n using Gaussian Elimination method.

int **gauss** (*mat,ans,nrow*)

double mat[][]	Coefficient matrix in the form $\mathbf{AX}=\mathbf{B}$, $[\mathbf{A} \mathbf{B}]$. The rank of the matrix is $N \times N+1$. If the argument is not a two dimensional array, then the total size of the array must be greater than or equal to $N*(N+1)$.
double ans[]	Array to hold the return answer, of which the size must be greater than or equal to N .
int nrow	Order of the set of equation, N . Note that if the matrix is given in two-dimensional array, the dimension must be $[N][N+1]$, then the presence of this argument is optional for checking only.
Return	0, if the system is unsolvable for a unique solution, and 1, if a unique solution is obtained.

This function applies the Gaussian Elimination method to solve the linear equation in the form:

$$\mathbf{AX} = \mathbf{B}$$

where \mathbf{A} is a $N \times N$ matrix and \mathbf{B} , $N \times 1$.

Note that there is no order limit to the set of equation. However, stack memory will be allocated for temporary matrix storage for fast operation if the order number of the matrix is less than or equal to 8. As for order number larger than 8, virtual memory will be used to cache the access to the matrix.

gcd

Intrinsic function to calculate the greatest common divisor.

Var **gcd** (*args...*)

Var args	Variable number of arguments of applicable data type. The minimum number of arguments are 2, while the maximum number are 15. Type conversions are performed automatically from left to right in sequence.
Return	The greatest common divisor, depending on the type of arguments.

This is an intrinsic function that calculates the greatest common divisor among arguments. Both upper case and low case of this function names are accepted. The type of operands can be integers, longs, and doubles. Type conversions from integer value to double are performed automatically when they are mixed in the operation.

Note that if the arguments are doubles, the calculation will be done in floating point. Since the calculation involves successive divisions, the truncation error in each floating point division may propagate through the whole calculation, and thus the accumulated effect of truncation errors may be significant, depending on the number of division required to complete the job.

get_dirlist

Obtain a listing of files from specific directory.

int **get_dirlist** (*path, match, result, fatrb*)

or (after V3.1.055)

int **get_dirlist** (*path, match, mid, fatrb, start*)

char path[] Path name of the target directory to get the file list. Null string or 0 means current directory.

char match[] Match pattern specification that follows the rule of **strmatch()** function to match the filename from the specific directory.

char result[][] A string array to hold the resulting list. The returned filename is at most 13 bytes in length including the terminating null. If the size of the string is smaller than a matched filename, the filename will be ignored.

If the size of the string is declared larger than 13 bytes, additional information will also be saved into the string buffer after byte 12, in the following order:

Byte 0-12: Filename, null terminated.

Byte 13: DOS file attribute flag. See **chmod()** and **fexist()** for file attribute.

Byte 14-17: File length, long integer.

Byte 18-19: DOS file creation time.

Byte 20-21: DOS file creation date.

int fatrb Optional file attribute to search for. Default 0 to search for all normal file. See **fexist()** for file attribute word definition.

int mid Memory text buffer handle returned by **tx_open()**, to store the returning result. Note that the returning result will have the full path name without the DOS file length and date/time informations. Use **filesize()** to obtain the file length.

long start Optional long value to specify the starting record number to store the filename into the memory text buffer, default to 0.

Return Number of files found and returned in the string array or the memory text buffer.

Note that the size of the string array will limit the returning result. TCL program may use **memmove()** or **setdata()** to access the file length and the other informations.

This function is supported in DOS version only of latest update.

get_fontinf

Obtain the information of a supported font file.

int **get_fontinf** (*fname, fontname, eid, index*)

char fname[] Full path name of the font file with explicit file extension. The file extension is checked first to determine the type of the file.

char fontname Optional string buffer to return the font name information read from the font file if it is available. If this argument is given, the font name information will be read and returned.

int <i>eid</i>[]	Optional integer array to hold the encoding ID of the fonts, if applicable and available.
int <i>index</i>	Optional integer to specify the TTF index in a TTC file, if the given font file is a TTC file. If the value -1 is given, it returns the total number of TTFs in the TTC file.
Return	Integer value indicating the major language type of the font, as described below: -1 Error, the file is not found. 0 Error, the file is not recognizable or not valid. 1 The file is an ASCII small font (SHX, PFB). 2 The file is a Unicode small font (SHX). 3 The file is a big font (SHX). 4 The file is a shape/symbol file (SHX). else Language code of the font. See os_language() for a table of the language code.

For TrueType Collection file (TTC), which contains multiple TrueType fonts in a file, the return value depends on the additional argument *index*. If this value is -1, this function will return the total number of TrueType fonts in the file; otherwise, the information from the TrueType font being indexed by the value will be returned.

For TrueType Font (extension 'TTF'), this function returns the language code read from the Microsoft Encoding Table contained in the file if one is found. If the TTF file does not contain any Microsoft Encoding Table, it return 0. The same is true if the file is a TrueType Collection file with a valid index value.

If the optional array *eid* is given and the font file is TrueType, **get_fontinf()** will return the encoding IDs read from the encoding table for the Microsoft Platform. There may be more than one encoding table, so there may be more than one encoding ID. A -1 will be stored at the end of the ID array to signify the end of the array.

get_profile Win

Read parameter string from specific profile file (INI file)

STRING get_profile (*section,entry,inifile*)

char <i>section</i>[]	The parameter section name.
char <i>entry</i>[]	The parameter entry name.
char <i>inifile</i>[]	Optional name of the profile file. If this argument is not given, it defaults to TwinCAD's current active profile file. If it is given without path, it defaults to the Windows directory.
Return	Empty string, if the specific entry, section or file is not found; otherwise, it returns the parameter string retrieved. The maximum length of the string returned is limited to 255.

The parameter string in the profile will be in the form as given below:

[Section]

Entry = String

....

See also **put_profile**.

get_syspath

Obtain path informations.

STRING **get_syspath** (*which*, *level*)

int which Specifies which path to return, optional, default 0. Valid numbers are as listed below:

- 1 The path from which the TCL program in query is loaded and executed. An optional integer argument **level** may be given to specify the level of the TCL program in query.
- 0 The path from which TwinCAD is running.
- 1 The Windows Directory returned by Windows (Windows Only).
- 2 The Windows System Directory returned by Windows (Windows Only).
- 3 The Windows Temporary Drive returned by Windows (Windows Only).
- else** Reserved, treated as 0.

int level Optional integer, default to 0, effective only when **which** is -1, to specify the level of the TCL program name to return, as described below:

- 2 The path name of the grand-parent TCL program that leads to the execution of the current TCL program.
- 1 The path name of the parent TCL program that runs the current TCL program.
- 0 The path name of the current running TCL program.
- 1 The path name of the latest child TCL program run by the current running TCL program.

Return Path information as required if applicable; otherwise, NULL string is returned.

getangle

Get user input of an angle value.

double **getangle** (*prompt*, *default*, *pbase*)

char prompt[] Optional character string as the prompting message.

double default Optional default angle in units of degree. If present, the function will accept user input of null return (space-bar) and return the default value; otherwise, the null return will be rejected.

POINT *pbase* Optional base point for angle dragging input. If present, it is taken as if the user has indicated the base point of an angle indication.

Return User input of an angle value in units of degree.

Besides the direct input angle value, this function will accept the angle specification by means of two points given successively. As soon as the user enters the first point (which may be given as a default from the function argument), a dragging line will be provided for the user to specify the second point or, of course, an angle value directly.

getblock

Get name of a block by means of pop window selection from the user.

String getblock()

Return Null string if user select nothing, else the name of the block selected by user.

This function will automatically pop up the block selection window as the system does for the BLOCK/INSERT command. However, this block name selection window does not allow the user to create a new block name.

getcdir

Return the current working directory of a specific drive.

STRING getcdir (*drvno*)

int *drvno* Drive ID number, 0:current drive, 1:A, 2:B, 3:C,... etc. optional, default to 0.

Return Null string if the drive number is invalid; otherwise, the full path name of the current working directory on the specific drive is returned.

getcolor

Pop color selection window and get a user selection of the color.

int getcolor (*color*)

int *color* Optional default color number (-1,0-15). If this argument is omitted or of wrong type, the default color will be the current active entity color (@CURCOLOR).

Return Integer value of the color selection. A -1 value means [ByLayer].

This function is used for color selection. It will pop up a color selection window as the one by the ECOLOR command.

getcorner

Get a user input of a corner point by dragging a box on the screen.

POINT getcorner (*pcorner,style*)

- POINT *pcorner*** The first corner point for the dragging box.
- int *style*** Optional line style of the dragging box. 0 for solid line, 1 for dotted line and 2 - 15 for different dashed line.
- Return** Another corner point input by the user.

This function will provide a dragging box with a corner fixed at the given point, for the user to specify the other corner.

getdist

Get user input of a distance value, accepting 2 points to determine the value.

double *getdist* (*prompt, default, pbase*)

- char *prompt[]*** Optional character string as the prompting message.
- double *default*** Optional default value for the user input. If this default value is omitted, then the user pressing the space bar or return key without entering anything will be rejected.
- POINT *pbase*** Optional base point for getting distance. If present, it is taken as if the user has indicated the base point of a distance indication.
- Return** Distance value as required.

Besides the direct input value, this function will accept the distance specification by means of two points given successively. As soon as the user enters the first point (which may be given as a default from the function argument), a dragging line will be provided for the user to specify the second point or a direct distance value.

gete

Obtain a single object from drawing database by prompting user for selection or by searching the display list directly at specific position.

ENTITY *gete* (*prompt, mask, cmask, ltmask*)

or

Entity *gete* (*where, mask, cmask, ltmask*)

- char *prompt[]*** Optional character string as prompting message.
- POINT *where*** Alternative argument of POINT data to specify pick point to the display list. This point must be within the current view window extent to be effective. See description text for details.
- int *mask*** Optional entity type mask, used to mask off undesired objects by entity type, default to 0xffff (accepting all type of entities).
- int *cmask*** Optional entity color mask, used to mask off undesired objects by their entity color, default to 0 (accept all colors).

int Itmask Optional entity linetype mask, used to mask off undesired objects by their entity linetype, default to 0.

Return Entity handle of the picked object. Return null if the user type <Ctrl/C> and **userbrk()** is off, or the user has pressed space-bar or return key without picking up an object. The application may use **userbrk()** to tell the cases.

Defined in the include file, tclcad.h, are the following macros:

```
#define FPOLY      0x01 /* Accept Polyline */
#define FPOINT    0x02 /* Accept Point */
#define FLINE     0x04 /* Accept Line */
#define FARC      0x08 /* Accept Arc, Ellipse */
#define FCIRCLE   0x10 /* Accept Circle */
#define FLINE3D   0x20 /* Accept 3d-line */
#define FTEXT     0x40 /* Accept Text */
#define FINSERT   0x100 /* Accept Insert */
#define FDIM      0x200 /* Accept Dimension */
#define FREGION   0x400 /* Accept Region */
#define FTAG      0x800 /* Accept Tag */
#define FSYMBOL   0x1000 /* Accept Symbol */
#define F3DFACE   0x2000 /* Accept 3DFace */
```

You may use these macros to setup the mask to indicate which kind of entities are allowed to be picked up from the drawing data base. If the mask is omitted, then all kinds of entities are accepted as a default. Note that if a segment of a polyline is picked up, it will return the segment's entity handle if the **FPOLY** is not setup; otherwise, it returns the polyline's entity handle.

If the first argument passed is not a string, but a data point, this function will directly query the system to pick the object at the specific position. If the pickup fails, the returned entity handle will be NULL. Successive pickups at the same point is also possible, provided no other **getxxxx()** function calls intervene. A few notes about this feature is given below:

- The pick point must be within the extent of the current view window; otherwise, it will pick up nothing.
- It is the **@PICKBOX** that defines the size of the internal target box to cross with the object, not the **@APERTURE**.
- If the pick point is the same as that from the last call to this function and there is no other interface function calls (including **command()**) in between, the system will assume that this call is for a successive pickup, which means it will search from the last matched entity exclusively. To avoid this case and to force a search from the beginning, you may issue a **command()** that does nothing or a false **gete()** function call with a different pick point first.

Note that once an object is picked, the system will automatically justify the pick point so that it will lie exactly on the picked object (at a nearest point).

The Color mask and the Linetype mask

After TCL runtime V3.02, this function supports the entity masking by color and linetype (as specified by the third and fourth arguments).

The color mask is an integer of 16 bits. Each bit of the mask from bit 0 (LSB) to bit 15 (MSB) corresponds to the color number from 0 to 15. If the corresponding bit flag is set to ON (1), the entity with the specific color number will be rejected.

The Linetype mask is also an integer of 16 bits. Each bit of the mask from bit 0 to bit 15 corresponds to a linetype ID from 0 to 15. If the corresponding bit flag is set to ON (1), the entity with the specific linetype ID will be rejected. Note that the linetype ID starts from 0 and the CONTINUOUS linetype's ID is always 0 (since it is the first linetype created).

Note about the object masking

The object masking provided by **gete()** is totally independent of the one provided in the Object Selection Operation. It will not be affected by the current SELMASK setting. That means, the layer masking will not be effective. And only being explicitly specified, will the **gete()** perform the object masking on the picked up object.

Example: To pick up a line, arc, or circle

```
e1 = gete("\nSelect objects: ",FARC|FLINE|FCIRCLE);  
• To pick up polyline only  
e1 = gete("\nSelect polyline: ",FPOLY);
```

getenv

Query and retrieve the content of a specific system environment variable.

STRING **getenv**(*varname*)

char *varname*[] Name of the variable to query and to retrieve.

Return Null string if the specific variable is not present; otherwise, the current content of the variable.

getesel

Obtain an entity selection list from the drawing database by prompting user to make the selection.

SLIST **getesel** (*prompt*, *mask*, *cmask*, *ltmask*)

char *prompt*[] Optional character string as prompting message.

int *mask* Optional entity type mask, used to set up the selection mask to mask off undesired objects by their entity type (defined in tclcad.h). It defaults to 0xffff (accepting all type of entities).

int *cmask* Optional entity color mask, used to mask off undesired objects by their entity color, default to 0 (accept all colors). See also description text.

int *ltmask* Optional entity linetype mask, used to mask off undesired objects by their entity linetype, default to 0 (accept all linytypes). See also description text.

Return Entity selection list containing the selected objects.

Use **ent_count()** to determine the number of entities in the selection list. Note that an entity selection list is a dynamically allocated array of ENTITY type. Use array subscript to access each element in the list. Be aware of that the lifetime of such dynamically allocated array is the same as the function body that contains the call to initialize it. When the function exits, it will be de-allocated automatically as if it were a local variable, regardless of whether the SLIST variable is declared in global scope or not.

See the description text of **gete()** for informations about the color masking and linetype masking.

SLIST **getesel** (*opr*, *mask*, *cmask*, *lmask*)

int opr Integer value to specify the default select operation.

-1 Use the previous selection set. The previous selection set will be checked out against the *mask* and returned as if the operator has entered the 'P' option and then ended the selection. If the optional masks are not given, it defaults to use current setup values.

-2 Select all entities. All the drawing entities will be checked out against the *mask* and returned as if the operator has entered the 'ALL' option and then ended the selection. If the optional masks are not given, it defaults to use current setup values.

0 and else Use system **SELECT** command prompt to select the entities. If the optional masks are not given, it defaults to accepting all entities without masking.

int mask Optional entity type mask, used to set up the selection mask to mask off undesired object by their types.

int cmask Optional entity color mask, used to mask off undesired objects by their entity color, default to 0 (accept all colors). See **gete()** for related details.

int lmask Optional entity linetype mask, used to mask off undesired objects by their entity linetype, default to 0 (accept all linytypes). See also **gete()** for related details.

Return Entity selection list obtained by the specific selection option.

If the current selection mask is NOT enabled, the above two automatic selection operations will not check the selected entities against the mask settings, even if these optional arguments are given, for reason of downward compatibility. These optional arguments are provided for your programming convenience to overwrite the system's current selection mask temporarily for the selection operation. Note that the layer masks are setup by **set_mask()** function.

SLIST **getesel** (*ent_st*, *mask*, *ent_nd*)

ENTITY ent_st Starting entity to search inclusively from the drawing database for the selection.

int mask Optional entity type mask, used to set up the selection mask to mask off undesired object by their types.

ENTITY *ent_nd* Optional ending entity to stop the searching inclusively from the drawing database for the selection. If this argument is not given, then the search starts from ***ent_st*** till the end of the drawing database.

Return Entity selection list obtained by searching the drawing database in the specific entity range.

This function call is useful in obtaining the newly creation of entities after a certain entity. For example, to obtain the newly created entities by a **command()** function call, obtain the last entity from the drawing database by **ent_last()** first before calling the **command()**, and use this function directly to obtain the selection list.

To use this function call with the color and linetype masking, set the system variables @COLORMASK and @LTYPEMASK directly. See also description text of **getel()** for details information on color and linetype mask.

SLIST **getesel (*ePoly*, 0)**

ENTITY *ePoly* Entity handle of a polyline from which all its component entities will be selected in sequence in the selection list.

Return Entity selection list obtained by placing the component of the given polyline entities.

This function call is supported only after version V3.1.058+; however, it will be accepted in previous version and always return an empty selection list. The TCL application can check against this fact to see if this function call is supported or not. An example program fragment is given below:

```
slist = getesel(ePoly,0);
if ( (n=ent_count(slist)) == 0 )
    // Do it in old way
    slist = ent_aloc(ent_count(ePoly));
    eCur = eFirst=ent_pnext(ePoly);
    for(n=0;ent_ok(eCur);n++)
        slist[n] = eCur;
        eCur = ent_pnext(eCur);
        if ( eid(eCur) == eid(eFirst) )
            break;    // Close poly!
    }
}
```

Example To erase all selected single segments

```
SLIST slist;
int i, ecount;
slist = getesel("\nSelect object: ",0xffff);
ecount = ent_count(slist);
for(i=0;i<ecount;i++)
    switch(ent_type(slist[i]))
        case T_LINE:
        case T_ARC:
            continue;
        default:
```

```

        slist[i] = ent_null();
        continue;
    }
}
command("ERASE \@slist ");

```

- Another way to make the selection

```

slist = getesel("\nSelect object: ",FLINE|FARC);

```

- Select all polylines from a layer

```

@selmask = ~FPOLY;           // Enable only polylines
@snapflag |= 0x2000;         // Enable Selection Mask
set_mask(0,"",0);           // Clear all Layer Mask
set_mask(0,"MYLAYER",1);    // Enable Layer: MYLAYER
slist = getesel(-2);        // Select All ...
...

```

- Select certain objects from the screen by crossing window

```

... // Setup masking if necessary
command("SELECT C P1 P2 "); // Do the selection
slist = getesel(-1);       // Select Previous

```

getfile

Pop up file window and obtain a file specification.

STRING **getfile** (*default,ext,iostate,titlestr*)

char <i>default</i>[]	Default of filepath and filename.
char <i>ext</i>[]	Default file extension, at most 3 characters.
int <i>iostate</i>	Integer flag to specify the type of file to get. <ul style="list-style-type: none"> Bit 0: 0 for input file, 1 for output file. Bit 1: 0, for free extension, 1 for fixed extension. Bit 7: 0 for R/W checking, 1, no R/W checking. Bit 13: 0, return DOS short filename, 1, return long filename if the long file name is supported (new feature after V3.1.058). Bit 14: 0, for ordinary files, 1, for drawing files with preview operation enabled. Bit 15: 0 for single file selection, 1, for multiple file selection, see descriptions. Else Reserved, should be zero.
char <i>titlestr</i>[]	Optional title string for the file window display title. If this argument is missing, the system default one will be used.
Return	Character string of user input of filename. Null string if user quit the file window.

Note that the file window will use the supplied path from the default filename from the argument for the initial drive-directory location setup. If there is any wildcat character in

the default filename, it also serves as the file mask in reading the directory; otherwise, it will be taken as the default filename and may appear in the filename entry field in the file window. However, if the *iostate* specifies to choose for the input file, the default filename will be checked for its existence. If it does not exist, the initial filename entry field of the file window will be clear without any default name, since the default is invalid.

The current drive and the current directory will be taken as the defaults, if the corresponding part in the default filename given as the first argument is missing. An empty string, thus, specifies the current drive and the current directory, without setting up any file mask.

Multiple File Selection

The bit 15 of the *iostate* is used to specify multiple file selection. If it is on, the File Window Operation will accept a multiple file selection operation. If there are files being selected when the user press the [OK] buttons, it returns the name of a temporary file that contains the list of the selected files. The TCL application may access the name of these selected files through the returning file, and may erase the file afterward.

The extension of the returned temporary file is always "FNL".

DOS Specific

Additional rules about Multiple File Selection are described below:

- All files from the file sub-window are un-selected as their initial states after they are read from the active directory. This means, whenever an operation causes the file window to read from the drive for a new list (even the same list) of filename, all files will be un-selected as an initial state.
- The basic operations are the same as those for the Single File Selection, except that once a file is picked, it's selection state will be toggled by once. A file in selected state will be displayed in different color and checked with a check mark (in case the display is in monochrome mode). The number of selected files is also displayed on top of the file sub-window after the file number.
- The operator may press function key **F1** to select all files from the file windows, and function key **F2** to deselect all of them.
- The operator may press Shift/⟨Gray Down Arrow⟩ key to select the file pointed by the keyboard cursor bar. The keyboard cursor bar will also advance by one entry. This is very helpful for the user to select a range of files by continuously pressing the Shift/⟨Gray Down Arrow⟩ key.
- The operator may press Shift/⟨Gray Up Arrow⟩ key to deselect the file pointed by the keyboard cursor bar. The keyboard cursor bar will also move up (backward) by one entry. This is very helpful for the user to deselect a range of files by continuously pressing the Shift/⟨Gray Down Arrow⟩ key.

Windows Specific

The Windows Common Dialog of File window is called for the operation.

getfsize

Obtain the size of a given file.

long **getfsize** (*fname*)

char fname[] Full pathname of the file to query.

Return File size in bytes. 0 if the file is not found or zero byte in length.

getinput

General input function from the user interface.

int **getinput** (*prompt,index,flag,keywords,...*)

char prompt[] Prompting message string.

int index TCL Register index number to hold the input.

int flag Optional control flag.

Bit 0 1, Pick Object Mode, 0, No pick object mode

Else Reserved, should be zero.

char keywords[] Optional keywords...

Return Integer value indicate the type of user input has received:

-1 User Breaks with <Ctrl/C>

0 User Presses Space-bar/Return

1 Value Input to **V[index]**

2 Point Input to **P[index],V[index]** (Z)

3 Object picked in **E[index]**, Point in **P[index],V[index]**

4 and up Option keyword selection (offseted by 4)

This function may return 3 (an object is picked) even if no object pick mode is enabled, as the operator may use object snap directive (ENDp, MIDp, etc.) to obtain the point. The picked point should be valid at **P[index]** as well as in the **.pick** field of the **E[index]**.

The register index value must be in the range from 0 to 19.

getkey

Wait and get user single key input from devices.

int **getkey**()

Return Integer key value with the keyboard scan code in high byte and ASCII equivalent in low byte.

This function will pause the execution of a TCL program until the user has input something from the keyboard, or from the button of the pointing devices, which is then returned as the result.

The result is a 2 bytes integer, with the high byte containing the standard IBM keyboard scan code and the low byte an ASCII equivalent. If the input is from the pointing devices, it is also converted into equivalent keyboard codes, as described below:

First button

Second button	Scan code as <Enter> key, lower byte is always 0xff.
Third button	Scan code as <Home> key, lower byte is always 0xff.
Fourth button	Scan code as <End> key, lower byte is always 0xff.
Pointer Movement	Scan code as <TAB> key, lower byte is always 0xff. Converted to <LeftArrow>, <RightArrow>, <UpArrow> and <DownArrow> keys as the pointer moves, with lower byte fixed at 0xff. Effective only in DOS version. In Windows version, the pointer is global to all Windows application and thus its movement is not trapped for the conversion.

For a table of keyboard scan codes, please refer to appendix.

getkeyword

Get user keyword input.

int **getkeyword** (*prompt,keywords...*)

char prompt[] Character string used as message prompt.

char keywords[] Maximum 15 arguments of acceptable keywords in character string. Note that the keywords string must not contain any space characters or control codes. The keywords are lower case sensitive. Use upper case characters to match both lower case and upper case input.

Return Integer order number from 0 to n, where n is the number of keywords. The return value of 0 indicates the user has pressed return key for default. The return value of 1 indicates the first keyword is specified by the user, the value of 2, the second keyword, and so forth.

The following rules must be noted in designing the keywords:

- **Use capital letters in the keywords as possible.** The input string will be checked first against these keywords for an exact match and, if there is no matching, it will be turned into upper cases and checked again. Therefore, if a keyword is in lower cases, the operator must enter the string in lower case and as exactly the same as the keyword to match it.
- **Use as fewer characters as possible in the keywords.** Single character keyword is the best, since it requires the operator to enter only one single key. You may have a lengthy description in the prompt, but declare the keyword as shorter as possible.
- **Should a keyword be a prefix to another, place the longer** one after the shorter one in the arguments. This is because the matching only requires the leading characters of the input string to match with the keywords.

The sub-command menu is enabled as specified in the system initial file, it will be popped up automatically to display the keywords. The display of keywords follows the rules for the sub-command menu. That is to say, a keyword will be displayed in double

width if there is enough room for it; otherwise, it will be displayed in single width and truncated if its length should exceed the maximum width of the display field.

getlayer

Get user selection of a drawing layer.

int getlayer (*deft*)

char *deft* Optional, Default layer ID.

Return ID of the layer selected by the operator from the popup GUI dialog window.

This function is used to obtain a user selection of a drawing layer via the popup GUI dialog window supplied by TwinCAD.

getln

Get user input of a line (two points).

LINE getln (*prompt, pbase*)

char *prompt*[] Optional character string as prompting message.

POINT *pbase* Base point of the line, used as the first given point of the line.

Return LINE from the given base point to user input point.

or

LINE getln (*prompt1, prompt2*)

char *prompt1*[] Optional character string used to prompt for the first point of the line.

char *prompt2*[] Optional character string used to prompt for the second point of the line.

Return LINE resulting from the user input. If the operator quits the input, the resulting line will be an invalid line.

There are two kind of calls to obtain a LINE specifications using this function. If the second argument passed is a POINT data, then it is taken as the first point of the line, and used as the base point in the line dragging, and thus only one more point is required of the user input. However, if the second argument is not a POINT data, then this function will require the operator to input two points, each with a prompting message taken from the arguments in sequence (provided they are of string type).

The operator may use the **TAN/PER** object snap directives in specifying the two points, as if he is drawing a line using the **LINE** command. When the second point of the line is in request, a dragging line is provided.

If the operator quits the input, the resulting line will be an invalid line. Use **ent_ok()** to test the returning line or use the **userbrk()** to test whether the operator has pressed the <Ctrl/C> or not.

GetLongName

Return long filename of a given file.

STRING GetLongName(*fname*)

char *fname*[] Optional short filename to check. If this argument is missing, this function will return the name of the long filename service if it is available.

Return NULL, if the long filename service is not available or the given file is not present. Otherwise, the long filename of the specified file will be returned.
If the argument *fname* is not given, this function will test and log in the current long filename service available. And it will return NULL, if there is no such service available; or it will return the name of the service.

This function is supported only after version V3.1.058+, which supports the long filename through an external DDE server called **LFNServer**.

getltype

Obtain linetype information from system

LTYPE getltype (*name*)

char *name*[] Optional name of linetype information to retrieve.

Return If the linetype of the specific name is located from the system table, the information of it will be returned. However, if the argument name is omitted or the specific linetype is not found, this function will return the first linetype entry in the system table, so that the subsequent information of the linetype can be retrieved via the **tbl_next()** function calls. To check whether the specific linetype exists or not, further checking on the returning information is required.

Or, to pop up the linetype selection window to get a user selection of the linetype, use the following function call:

LTYPE getltype (*id*, *type*)

int *id* Default linetype ID of the user selection.

int *type* 1, linetype only, 0, include [ByLayer] in the selection list.

Return If the name of the return linetype data is NULL, it means the user has chosen [ByLayer]; otherwise, it is the chosen or default one returned.

This function call is supported only after V3R3b.

Example: To test if a linetype is present or not

```
if ( getltype("CENTER").name=="CENTER" )
    printf("\n'CENTER' line is present");
else
    printf("\n'CENTER' line is not present");
```

getp

Get user input of a point.

POINT **getp** (*prompt,default*)

char *prompt*[] Optional character string as prompting message.

POINT *default* Optional default point for user null input. If this argument is omitted, system will reject the user input of pressing space bar. Note also that this default point is served as the last point coordinate for the user point input in a temporary base.

Return Point that the user has just input.

getpath

Invoke the Directory Browser and to return a user selection of drive/path.

STRING **getpath** (*drvpath,explevel,titlestr*)

char *drvpath*[] Optional initial default drive/path for the selection, default to current directory on current drive.

int *explevel* Optional automatic expanding level, default 0.

char *titlestr*[] Optional title string for the Directory Browser.

Return The full pathname of the drive/path selected or entered by the user. If the user quit the selection operation, it returns null string.

GetRGBColor Win

Get user selection of an RGB color value.

long **GetRGBColor** (*deft,usr*)

long *deft* Optional initial default RGB color for user to specify, default 0.

long *usr*[] Optional, customer RGB colors (at most 16) to be used for user selection.

Return RGB color value that user has specified.

This function calls the **Windows's Common Dialog API ChooseColor()** for an RGB color selection.

The RGB color is a 24-bit color value stored in a long integer word. Each RGB component takes up 8 bits. The R-component takes up the lowest 8 bits, G-component, the next 8 bits, the B-component, the next higher 8 bits. The highest 8 bits are not used and should be 0.

gets

Get user input of a character string.

STRING gets (*prompt,nchar,default*)

char *prompt*[] Optional character string as prompting message.

int *nchar* Optional integer to specify the maximum number of character to get. The default is 72.

char *default*[] Optional character string used as the default input string.

Return A character string of user input.

Note this function will open a text entry field at the end of the prompt with the given number of column in width. User of DOS version may invoke the external input handle by pressing <Ctrl/Fn> as does for the text input to the TEXT command. The <Ctrl/Fn> function is also supported in Windows version but for text input handler written in TCL only.

GetShortName

Return DOS short format filename from a long filename.

STRING GetShortName(*fname*)

char *fname*[] Optional long filename to check. If this argument is missing, this function will return the name of the long filename service if it is available.

Return NULL, if the long filename service is not available or the given file is not present. Otherwise, the DOS short format filename of the specified file will be returned.

If the argument *fname* is not given, this function will test and log in the current long filename service available. And it will return NULL, if there is no such service available; or it will return the name of the service.

double **getv** (*prompt, default*)

char *prompt*[] Optional character string as prompting message.

double *default* Optional default value.

Return Double value of user input as required.

Note that **TCL** will automatically convert double value to integer value and vice versa.

gextent

Get entity argument's geometry extent

LINE **gextent** (*ent, ext*)

Var *ent* Argument of entity type, such as POINT, LINE, ARC, CIRCLE, ENTITY handle or Selection list. String argument is also accepted to pre-evaluate the extent of the TEXT string under current default setup (style, height, width, etc.).

LINE *ext* Optional initial extent specification. If this argument is present and valid, the resulting geometry extent will be superimposed with this initial extent specification, useful for successive geometry extent finding.

Return Resulting geometry extent in the form of a LINE, from minimum geometry extent point to the maximum extent point.

Note that the calculation of entity's drawing extent will be affected by **set_trns()**. See **set_trns()** function for more information on 2-D transformation.

This can be used to determine the geometry relationship between an object and a given line. For example, to determine whether a polyline is at the right side or left side of a line, or is crossing the line, simply set the rotation base point at the start point of the line and rotation angle with the direction of the line, and calculate the polyline's extent. If the minimum Y extent is greater than zero, it is at the right side of the line and the value is the distance. If the maximum Y extent is less than zero, it is at the left side of the line, and the value is the distance. Otherwise, it is across the line, and the Maximum and Minimum Y extents tells how the line is dividing the polyline.

To determine the geometry relationship between an object and a given circle (ARC), use **pe_dist()** with the center point. Compensate the resulting distance value by the radius value. If the object is a polyline, positive distance means the point is to the right of the polyline, and negative means to the left. If the absolute value of the returned distance is less than the radius of the circle, it means the circle intersects with the polyline.

If the function is called with no argument at all, this function will return the total extent of the current drawing file. This is a very helpful feature.

Note that this function also accepts ENTDATA argument. However, it will not trap on invalid ENTDATA but ignore it and return invalid drawing extent value. This feature is supported only after TwinCAD V3.1.052. TCL application may check to see if the function support of **ent_init()** is present or not before taking advantage of this feature.

gr_screen DOS

Screen image I/O operation function

int gr_screen (*cmd*, *fname*)

- int *cmd*** Command code, see below:
- 0** Save current screen to file in **TCAM/Graphic** Runtime's private format. No file extension is assumed.
 - 1** Restore current screen from the file previous written by **TCAM/Graphic Runtime**.
 - 2** Load TCAM SCN file to screen. The TCAM SCN file can be produced by TCAMSCN.COM utility.
 - Else** Reserved for future expansion.
- char *fname*[]** File name for the I/O operation.
- Return** -1, if the specific file is not found or unable to create it, and 0 if it seems Ok!

The TCL program may fork out external executive program to run under the graphic mode by the **exec()** function. However, care must be taken by the TCL programmer not to alter the screen after the program exits. The TCL interpreter is in no way able to restore the graphic screen if a TCL program has ruined it. If necessary, the TCL programmer may use this function to save the screen first before doing the **exec()** and restore it back later.

The screen image file produced by this function call is used to backup the graphic screen temporarily for later screen restoration. The file format is subjected to change upon the different version of Graphic Runtime for different type of graphic display.

This function is no longer supported in Windows version.

GUIColor Win

Get/Set TCAM GUI palette color.

long GUIColor (*which*, *color*)

- int *which*** TCAM GUI palette color number, ranged from 0 to 15, or status palette color ranged from 16 to 31.
- long *color*** Optional RGB color to re-define the specified TCAM GUI palette color. If this argument is not given, the specified palette color will not be re-defined.
- Return** The RGB color of the specified TCAM GUI palette. If the palette color is to be re-defined, this gives the last RGB color of the palette.

All the GUI windows created by TCL function calls as well as those created by the TwinCAD commands that are compatible with its DOS counterpart are of special window class called TCAMGUI, which is simulating the TCAM GUI used in DOS version. The TCAMGUI assumes the display is a standard VGA using 16 color palette. This function can be used to modify the actual color palette in use.

The status palette colors (number from 16 to 31) are added after V3.1.055 and are use for the following purpose:

- Color 16: Not used, reserved, default to RGB(0,0,0).
- Color 17: X-Hair Cursor, default to RGB(192,192,192).
- Color 18: Dragging object,default to RGB(192,192,192).
- Color 19: Target box, default to RGB(192,192,192)
- Color 20: Snap box, default to RGB(192,192,192)
- Color 21: Axis Icon, default to RGB(192,192,192)
- Color 22: Blip mark, default to RGB(192,192,192)
- Color 23: Object Hightlight, default to RGB(192,192,192)
- Color 24: Not used, default to RGB(128,128,128)
- Color 25: Dotted Hightlight, default to RGB(0, 0 ,255)
- Color 26: Not used, reserved
- Color 27: Not used, reserved
- Color 28: Not used, reserved
- Color 29: Not used, reserved
- Color 30: Selection fence, default to RGB(255,255,0)
- Color 31: Not used, reserved, default to RGB(255,255,255)

GUIFont Win

Setup the current variable text font for the subsequent text output via **wnd_puts()**.

int GUIFont (*height, face, charset, pit, wflag, outprec*)

int *height* Optional text height in logical unit, specifying to change to current text font height. If it is 0, it will reset all text font parameters to Windows internal defaults.

char *face*[] Optional name string to change the name of the current text font face.

int *charset* Optional integer to change the character set. Interested character set values are given below:

```
#define ANSI_CHARSET          0
#define DEFAULT_CHARSET      1
#define SYMBOL_CHARSET       2
#define SHIFTJIS_CHARSET     128
#define HANGEUL_CHARSET     129
#define CHINESEBIG5_CHARSET  136
#define OEM_CHARSET          255
```

int *pit* Optional integer to change the pitch and family. The lower 4 bits are used to specify the pitch, and the next higher 4 bits are used to specify the font family, as the definitions given below:

```
/* PitchAndFamily pitch values (lower 4 bits) */
#define DEFAULT_PITCH        0x00
```

```
#define FIXED_PITCH      0x01
#define VARIABLE_PITCH  0x02
/* PitchAndFamily family values (higher 4 bits) */
#define FF_DONTCARE     0x00
#define FF_ROMAN        0x10
#define FF_SWISS         0x20
#define FF_MODERN       0x30
#define FF_SCRIPT        0x40
#define FF_DECORATIVE   0x50
```

int *wflag* Optional integer to change the weight and font options. The value of the lower 12 bits are used to specify the font weight. A zero value means "Don't care". Interested weight values are given below:

```
/* wflag weight values */
#define FW_DONTCARE      0
#define FW_THIN          100
#define FW_EXTRALIGHT   200
#define FW_LIGHT         300
#define FW_NORMAL        400
#define FW_MEDIUM        500
#define FW_SEMIBOLD      600
#define FW_BOLD          700
#define FW_EXTRABOLD     800
#define FW_HEAVY         900
```

Bit 12 to bit 15 of *wflag* are used for font options:

- Bit 12:** Reserved, should be zero.
- Bit 13:** Strike-out text options.
- Bit 14:** Underline text options.
- Bit 15:** Italic text options.

int *outprec* Optional integer to change the output precession setting, which can be one of the following values:

```
#define OUT_DEFAULT_PRECIS  0
#define OUT_STRING_PRECIS   1
#define OUT_CHARACTER_PRECIS 2
#define OUT_STROKE_PRECIS  3
#define OUT_TT_PRECIS       4
#define OUT_DEVICE_PRECIS   5
#define OUT_RASTER_PRECIS   6
#define OUT_TT_ONLY_PRECIS  7
```

Return Current text font height in logical unit.

This function is used to setup a logical font data as the the font used in the subsequent text output via **wnd_puts()** function call. **TwinCAD** will call Windows API CreateFontIndirect() to create the font upon these given parameters for the text font display via **wnd_puts()**.

See Windows SDK document for the meaning of these values listed in the macro definitions.

i or I

Intrinsic function to convert double value to integer.

int **i** (*Argument...*)

Var Argument Variable number of arguments of predefined type. See **v()** or **V()** for the argument formats.

Return Integer value of the evaluated result. The conversion is done by rounding the double toward zero (direct truncation of fractional part). If the conversion overflows, the largest value with the same sign is returned. If the conversion underflows, then zero is returned.

This is a Type Casting Function (Register Function) in corresponding to the I registers. It accepts the same arguments format as **v()** or **V()**, but returns the result in integer type.

identify

Identify the type and existence of an identifier, named object, module, or anything that can be identified and referenced by name string.

int **identify** (*name*)

char name[] Name of object to identify

Return Integer value indicating the type of the name, as below:

- 0** The name does not exist.
- 1** The name is a kernel Intrinsic function.
- 2** The name is a TCL local function.
- 3** The name is a loaded FUNCTION.
- 4** The name is a loaded COMMAND.
- 5** The name is a TCL Runtime function.
- 6** The name is a User Variable.
- 7** The name is a System Variable.
- 8** The name is a Prime Command.
- 9** The name is a an External Command.
- Else** Reserved.

In case that a name is overloaded, this function will return its effective type (the most outstanding use of the name currently). If the name is prefixed with a '@', then it is checked against the system variables and user variables only.

A loaded COMMAND is also a loaded FUNCTION, except that the latter requires exact match of letter cases of name.

This function is useful in determining whether a specific TCL program has been loaded or not, or a specific runtime function or command is supported by the version running the program.

Windows Specific

This function will return **9** if the given name is a registered **External Command**. A registered external command is a command defined by the name of a TCL program file located at the **TwinCAD's** COMMANDS sub-directory. External commands are registered at start-up time. But if new TCL program files are copied to the COMMANDS sub-directory during **TwinCAD** session, their names will be effective as external commands, though they are not registered yet and can not be identified by this function.

ifix

Intrinsic function to convert double value to integer.

int **ifix** (*value*)

double value Double value to be converted into integer.

Return The integer conversion result. The conversion is done by rounding the double toward zero (direct truncation of fractional part). If the conversion overflows, the largest value with the same sign is returned. If the conversion underflows, then zero is returned.

This function is added after V1.03 to replace the obsoleted **int()** function. Since **int** is a reserved word in TCL, the use of the **int()** must be in capital letters. The existence of the **int()** is of historical reason, and the use of it in TCL programming is not recommended.

in_span

Test whether a given point is in the span of a given entity.

int **in_span** (*ent,pnt*)

Var **ent** Variable type of LINE, ARC, or ENTITY handle that points to a line or arc.

POINT pnt Point data to test.

Return 0, if the test point is not in the span of the given geometry element, and 1, if it is.

Note that it is not necessary that the data point be exactly on the given geometry element to be in the span of it. With respect to a line segment, the data point is tested to see if its coordinates fall within the ranges confined by the two end points on either of the two axes. As for an arc segment, it is tested to see whether a line from the center of the arc in the direction toward the data point intersects with the arc or not.

To test whether a given point is exactly on the specific entity, apply **pe_dist()** after **in_span()** test. If the **pe_dist()** also return 0, the point lies on the specific entity.

index

Intrinsic function to locate a sub-string from a given string.

int **index** (*s1,s2*)

char s1[] Source string, null terminated.

char s2[] Sub-string to find, null terminated.

Return 0, if the sub-string not found, else the index position of the sub-string in the source string counted from 1.

inp

Input a byte directly from a hardware port.

int **inp(port)**

int port I/O port address.

Return Single byte read (by "IN AL,port" instruction) from the port is returned in low byte.

Note that this function will not work on NT machine.

inpw

Input a word directly from a hardware port.

int **inpw(port)**

int port I/O port address.

Return Two bytes read (by "IN AX,port" instruction) from the port are returned.

Note that this function will not work on NT machine.

is_running Win

Check to see if a shelled program is still running.

int **is_running (pid)**

int id The program ID returned by **winexec()** function call.

Return True if the shelled program is still running in the system, False if it had been terminated.

kbdstate

Sense the current state of keyboard from the BIOS call.

int **kbdstate()**

Return Status of keyboard, details as below:

Bit 0 Right shift key held down

Bit 1 Left shift key held down

- Bit 2** Ctrl key held down
- Bit 3** Alt key held down
- Bit 4** Scroll lock key active
- Bit 5** Num lock key active
- Bit 6** Caps lock key active
- Bit 7** Insert state locked active
- Bit 8-15** Reserved

Windows Specific

This function will always return 0 in Windows version.

keyin

Test and read the user input without waiting,

int **keyin()**

Return 0, if no user input, else scan code of user input, same as **getkey()**.

This function is effective only when user break function is off by `userbrk(0)`. The <Ctrl/C> input will be read as a data key input by this function. No special processing will be done on that.

I or L

Intrinsic function to return a line by geometry definition.

LINE **L** (*argument...*)

Var argument Variable number of arguments of predefined type. Different number of arguments of different type will specify different definition of a line.

Return Line of the definition results.

There are many ways to define a line. To provide each of them with separate functions is cumbersome in these functions's naming as well as in referencing them. Recalling that the operator may directly enter a valid TCL expression to the command line and utilize the power of TCL expression to facilitate his drafting operation, lengthy naming would certainly be unacceptable. This is one of the major reason to provide a general line geometry definition function, **I() or L()** (use upper case would be better).

The methods used to define a line depends on the arguments passed to the function, and are described below:

- ent** **Define a line by directly loading it from an entity** handle or another line entity. Valid argument types are LINE, and ENTITY.
- ps,pe** **Define a line by directly giving two points.** The two arguments must be of POINT type. Note that the z-coordinates of the two end points of the line will be assumed as the current entity elevation (**@elevation**).

- ps,pe,z** **Define a line by directly giving two points and an elevation value.**
 The first two arguments must be of POINT type, while the third argument, the z-coordinate value for the two end points of the resulted line.
- ps,pe,zs,ze** **Define a 3D line by explicitly giving two points and two z-coordinate values.** The first two arguments must be of POINT type, while the third and fourth arguments, are the z-coordinate values for the two end points of the resulted line, respectively.
- ps,dx,dy** **Define a line by giving one start point** (the first argument of POINT type), **and two delta values each in X and Y directions**, respectively, for the ending point. The z-coordinates of the two end points of the resulted line will be assumed as the current entity elevation (**@elevation**).
- ps,len,(ang)A** **Define a line by giving one start point** (the first argument of POINT type), **the length of the line** (the second argument of double), **and the direction angle of the line** (the third argument of double, tagged with 'A') in units of degree. The z-coordinates of the two end points of the resulted line will be assumed as the current entity elevation (**@elevation**).

Note that this intrinsic function will be invoked automatically by the pure assignment operator (=) when the data type of the L-value is LINE. For example, the expression

L1 = P1,P2

will be equivalent to the expression

L1 = L(P1,P2)

where the **L()** function is called by the assignment operator to evaluate the comma expressions to the right of it, which define a line from the position stored in P1 to that in P2.

l_cctan

Find the specific common tangent line of two given circle.

int **l_cctan (c1,c2,which,result)**

CIRCLE c1,c2 Circles of which the common tangent line is to find.

int which Integer code to specify which common tangent line to return. Only two bit fields are tested for the specifications:

Bit 0: 1, if the tangent point at circle C1 is to the left of the line passing through both center, and 0, if it is to the right.

Bit 1: 1, if the tangent point at circle C2 is to the left of the line passing through both center, and 0, if it is to the right.

The direction of the center line is taken from the center of the first circle C1 to the second circle C2.

LINE *result Pointer to a line where the result is to be returned.

Return Integer, 0 if there is no such answer, and 1 if ok.

If both circle intersects, there would be no inner common tangents. If either one is contained in another one, there would be no answers for neither outer nor inner common tangents. Note that there is also no answer for the trivial case when the circles are tangent to each others and the line passes through the point of tangent, since a line segment must be ended with two distinct points.

layer

Create or modify a layer.

LAYERlayer (*name,ltype,color,state*)

char *name*[] Name of the layer, leading space are removed, with a Maximum 12 characters.

char *ltype*[] Name of linetype to associate with the layer. Null string for default. If the specific ltype is not present, the "CONTINUOUS" will be assumed.

int *color* Integer color value associated with the layer. Negative value for default.

int *state* Integer flag to specify the layer state.

or

LAYERlayer (*ldata*)

LAYER *ldata* Layer information in a LAYER structure.

or

LAYERlayer (*id*)

int *id* Layer ID number from 0 to 255, specifying which layer information to retrieve by ID number.

Return The resulting layer information read-back. If the call is unsuccessful, due to bad layer name, the returning layer will contain null string in its name.

If the specific layer is present, this function call will modify the layer with the specific property value (linetype, color, etc). If it is not present, this function call will create it. After the function call, the specific layer will become current layer. Note that the layer name will be converted to upper case.

Special Layer Function Call

void layer (-1)

This function call will save all the layer's status to a status buffer, so that they can be restored back at later time.

void layer (-2)

This function call will restore all the layer's status from the layer status buffer.

Note that there is only one level of layer status buffer. There is no stacking mechanism provided for this save and restore operation. Again, like the **layer_on()** and **layer_off()** functions, this function call will not cause regeneration of the drawing.

Example: To create a layer name "BULE" with continuous linetype:

```
LAYER mylayer;  
mylayer = layer("BLUE", "", 1, 0);  
• To change a layer's color:  
mylayer.color = 2;  
layer(mylayer);
```

layer_off

Directly turn specific layer OFF.

int **layer_off** (*namepatn*)

char *namepatn*[] Match pattern of layer name

Return Number of layers has been changed from ON to OFF.

See also **layer_on()** description.

layer_on

Directly turn specific layer ON.

int **layer_on** (*namepatn*)

char *namepatn*[] Match pattern of layer name

Return Number of layers has been changed from OFF to ON.

The **layer_on()** and **layer_off()** functions are provided for fast layer ON/OFF control on all or selected layers. For example, to turn all the layers ON, issue **layer_on("*")** directly from the command line will do the job.

Note that these functions modify the state of layers directly, and will not cause regeneration of drawing database immediately. The operator or the TCL program must issue **REGEN** command to regen the drawing.

See **strmatch()** for the use of wildcat character in the match pattern name.

layerid

Obtain a layer's internal ID number.

int **layerid** (*name*)

char *name*[] Name of the layer, leading space are removed, with a Maximum 12 characters.

Return The internal ID number of the specific layer, from 0 to 255. If the specific layer is not found, or the argument is invalid, it return -1.

To alter an existing entity's layer is to change its internal assignment of layer ID number. Note that if you deliberately alter an entity by its layer ID number with one that is not yet valid (i.e., no layer being associated with it), a layer with default attributes (color 0, continuous linetype, non-frozen state) is assumed.

Also note that the internal ID number for a layer is not necessarily the same as the number implied by its name, though the system may try to assign such one as the implied as possible; nevertheless, there is at least one layer for sure to be so assigned, which is the layer "0". The layer "0" is always created with an internal ID number of 0.

left\$

Intrinsic function to take a part of a string from its beginning.

STRING left\$ (*str*,*n*)

char *str*[] Source string.

int *n* Number of characters from the beginning of the string to take.

Return Sub-string taken from the first argument up to the specific number of characters from its beginning. If *n* is larger than the length of the source string, then the whole string is returned.

len

Intrinsic function to return the length of a string

int len (*str*)

char *str*[] String to find length, null terminated.

Return The length of the string argument. **0** means a null string.

This function implements the same function as **strlen()** in standard C library, but with a different name for historical reason. For portability consideration, use the **strlen()** function which may be defined by the following macro:

```
#define strlen(s1) len(s1)
```

The include file `tblstr.h` contains several macro definition for standard C string functions.

linetype

Create, modify a linetype

LTYPE linetype (*name*,*unit*,*dash*)

char *name*[] Name of the linetype, leading space are removed, with a Maximum 12 characters.

double *unit* Basic unit value of the dash-dot sequence.

int dash[12] Integer array specifying the dash-dot sequence.

or

LTYPE linetype(ltdata)

LTYPE ltdata Linetype information in a LTYPE structure.

or

LTYPE linetype (id)

int id Linetype ID number from 0 to 255, specifying which linetype information to retrieve by ID number.

Return The resulting linetype information read-back. If the call is unsuccessful, due to bad linetype name, the returning linetype will contain null string in its name.

If the specific linetype is present, this function call will modify the linetype with the specific property value (unit, dash-dot). If it is not present, this function call will create it. Note that the linetype name will be converted to upper case.

In

Intrinsic function to calculate the natural logarithm function.

double ln (value)

double value Positive double value.

Return The natural logarithm of the argument value.

or

POINT ln (z)

POINT z Complex number.

Return The natural logarithm of the complex argument. The result is also a complex number.

lower

Convert argument string to lower cases.

STRING lower (str)

char str[] String argument to convert to lower case.

Return A temporary copy of the result (used for assignment).

ltypeid

Obtain a linetype's internal ID number.

int **ltypeid** (*name*)

char name[] Name of the linetype, leading space are removed, with a Maximum 12 characters.

Return The internal ID number of the specific linetype, from 0 to 255. If the specific linetype is not found, or the argument is invalid, it return -1.

To alter an existing entity's linetype is to change its internal assignment of linetype ID number. Note that if you deliberately alter an entity by its linetype ID number with one that is not yet valid (i.e., no linetype being associated with it), the CONTINUOUS linetype is assumed.

log

Intrinsic function to calculate the base-10 logarithm function.

double **log** (*value*)

double value Positive double value.

Return The base-10 logarithm of the argument value.

Note that this function calculate the base-10 logarithm, not the natural logarithm as in the standard C library. It is the **ln()** that calculate the natural logarithm.

max

Intrinsic function to return the largest values from arguments.

Var **max** (*args...*)

Var args Variable number of arguments of scalar data type (integers, longs, doubles). The minimum number of arguments are 2, while the maximum number are 15.

Return The largest one from the arguments. The returning type depends on the arguments.

memmove

Move data from one place to another in ANSI standard call convention.

int **memmove** (*dest, source, nbytes*)

var *dest Pointer expression of the destination variable.

var *source Pointer expression of the source variable.

int *nbyte* Optional number of bytes to move. This argument is optional but is recommended to supply explicitly.

Return Number of bytes of data actually moved.

If the number of bytes to move is larger than the destination variable can hold, error will occur.

The system variable with an array informations can be passed as an argument for the data moving operation. This is also the only way for a TCL application to access such system variable.

Note: BE AWARE OF THE DIRECTION. The first argument is the destination variable and the second is the source.

memset

Fill data memory with a specific byte content in ANSI standard call convention.

int **memset (*dest*, *cc*, *nbytes*)**

var **dest* Pointer expression of the destination variable.

char *cc* Character byte to fill into the destination memory.

int *nbyte* Optional number of bytes to move. This argument is optional but is recommended to supply explicitly.

Return Number of bytes filled in the memory.

If the number of bytes to fill is larger than the destination variable can hold, error will occur.

menucmd

Execute a inserted menu script command.

void **menucmd (*arglist...*)**

arglist At most 16 arguments can be given to this function call. Only arguments of string type, numerical value and POINT type are accepted. Numerical values (integer or floating point) and POINT type values are converted into equivalent strings. These strings are inserted to simulate the command script input to the system command prompt. Each argument is responsible for such command script input line.

This function is used to create a temporary script and then submit it to the system kernel for execution as if the script was from a menu file. This function differs from the **command()**

will issue the command script "Break ..." as a menu script which will be executed over and over until the operator has pressed <Ctrl/C> to stop it. Note that the pair of '[' is necessary as a part of menu item syntax. Note: The use of ^C syntax in the script will stop this feature.

MessageBox Win

Issue system message box

int **MessageBox** (*msg,title,type*)

char msg Main message string, can contain multiple lines delimited by '\n' characters.

char title Optional title string.

int type Specifies the type of the message box, default to 0.

Return The following standard dialog button ID code will be returned if the operator has depressed the corresponding button from the message box:

- 1 IDOK, user has pressed **OK** button.
- 2 IDCANCEL, user has pressed **Cancel** button.
- 3 IDABORT, user has pressed **Abort** button.
- 4 IDRETRY, user has pressed **Retry** button.
- 5 IDIGNORE, user has pressed **Ignore** button.
- 6 IDYES, user has pressed **Yes** button.
- 7 IDNO, user has pressed **No** button.

This function call **Windows API MessageBox()** directly. See Windows SDK document for details of the function.

The following definitions apply to the argument *type*.

```
// Button usage specifications
#define MB_OK                    0x0000 // OK button, default
#define MB_OKCANCEL            0x0001 // OK, Cancel buttons
#define MB_ABORTRETRYIGNORE 0x0002 // Abort, Retry, Ignore
buttons
#define MB_YESNOCANCEL        0x0003 // Yes, No, Cancek buttons
#define MB_YESNO            0x0004 // Yes, No buttons
#define MB_RETRYCANCEL        0x0005 // Retry, Cancel buttons
// Icon specifications
#define MB_ICONSTOP            0x0010 // Stop sign
#define MB_ICONQUESTION        0x0020 // ? sign
#define MB_ICONEXCLAMATION    0x0030 // ! sign
#define MB_ICONINFORMATION    0x0040 // Information ...
// Modal status
#define MB_SYSTEMMODAL        0x1000
```

```
#define MB_TASKMODAL    0x2000
```

This function call **Windows API MessageBox()** directly. See Windows SDK document for details of the function.

min

Intrinsic function to return the smallest values from arguments.

Var **min** (*args...*)

Var args Variable number of arguments of scalar data type (integers, longs, doubles). The minimum number of arguments are 2, while the maximum number are 15.

Return The smallest one from the arguments. The returning type depends on the arguments.

mkdir

Create a new directory with a specified name.

int **mkdir** (*dirname*)

char *dirname*[] Pathname of the directory to create.

Return 0 if the operation is successful, else the DOS error code:

- 2 File not found.
- 3 Path not found.
- 5 Access denied (if a file or directory with the specified name already exists in the specified path).

mod

Intrinsic function to implement the '%' (remainder) operation.

Var **mod** (*args...*)

Var args Variable number of arguments of applicable data type. The minimum number of arguments are 2, while the maximum number are 15. Type conversions are performed automatically from left to right in sequence.

Return Result of the remainder, depending on the type of arguments.

This is the intrinsic function to implement the '%' operation in the expression. In other words, the expression: `a%(b%(c%...` can be written as `MOD(a,b,c,...)`. Note that both upper case and low case of the function names are accepted. The type of operands can be integers or doubles. Type conversions from integer value to double are performed automatically when they are mixed in the operation. table below:

```
Operand1 - integer double
Operand2 | -----
integer  | integer double
```

double | double double

mul

Intrinsic function to implement the '*' (multiplication) operation.

Var mul (args...)

Var args Variable number of arguments of applicable data type. The minimum number of arguments are 2, while the maximum number are 15. Type conversions are performed automatically from left to right in sequence.

Return Result of the multiplication, depending on the type of arguments.

This is the intrinsic function to implement the '*' operation in the expression. In other words, the expression: a*b*c*... can be written as MUL(a,b,c,...). Note that both upper case and low case of the function names are accepted. The type of operands can be integers, doubles, or POINTs. Type conversions from integer value to double are performed automatically when they are mixed in the operation. See the table below:

Operand1 -	integer	double	POINT
Operand2	-----		
integer	integer	double	POINT
double	double	double	POINT
POINT	POINT	POINT	POINT

Multiplying a POINT data by a scalar value produces a scaling effect over the POINT with respect to the origin. Note also that a POINT data is also taken as a complex number.

mv_trns

Perform axonometric transformation on given entity.

Var mv_trns (ent,z)

Var ent Single entity to be transformed. Valid entities are POINT, LINE, ARC, CIRCLE, and ELLIPSE, or ENTITY handle pointing to these types of entity.

double z Optional elevation of the entity to be transformed. The default value is zero. This argument will overwrite the 3-D components of the LINE data, if it is given.

Return The transformation result of the same data type as the first argument is returned, excepts the ARC and CIRCLE data. It is always the ELLIPSE data returned for the ARC and CIRCLE data after transformation.

newent

Create elementary entities, such as points, lines, arcs and circles, to the drawing database.

ENTITY newent (entlist...)

Var entlist At most 16 arguments of basic entities such as POINT, LINE, ARC and CIRCLE.

Return The handle of the first created entity is returned.

The creation of new entity will be governed by the current setting of layer, color, linetype, elevation and thickness. TCL programming may access these settings by the system variables: **@clayer**, **@curcolor**, **@curltype**, **@elevation**, and **@thickness**.

This function also accepts ENTDATA argument to create the required drawing entity. However, it will not trap on invalid ENTDATA but ignore it and return NULL entity. This feature is supported only after TwinCAD V3.1.052. TCL application may check to see if the function support of **ent_init()** is present or not before taking advantage of this feature.

os_language Win

Detecting the language used by the operating system

int os_language()

Return The language ID code used by the operating system. Possible values are given in the table below:

Code	Language	Used in Country
0x0401,	Arabic,	Arab Countries
0x0402,	Bulgarian,	Bulgaria
0x0403,	Catalan,	Spain
0x0404,	Traditional Chinese,	R.O.C., Taiwan
0x0804,	Simplified Chinese,	People's Republic of China
0x0405,	Czech,	Czechoslovakia
0x0406,	Danish,	Denmark
0x0407,	German,	Germany
0x0807,	Swiss German,	Switzerland
0x0408,	Greek,	Greece
0x0409,	US English,	United States
0x0809,	UK English,	United Kingdom
0x040a,	Castilian Spanish,	Spain
0x080a,	Mexican Spanish,	Mexico
0x0c0a,	Modern Spanish,	Spain
0x040b,	Finnish,	Finland
0x040c,	French,	France
0x080c,	Belgian French,	Belgium
0x0c0c,	Canadian French,	Canada
0x100c,	Swiss French,	Switzerland
0x040d,	Hebrew,	Israel

0x040e,	Hungarian,	Hungary
0x040f,	Icelandic,	Iceland
0x0410,	Italian,	Italy
0x0810,	Swiss Italian,	Switzerland
0x0411,	Japanese,	Japan
0x0412,	Korean,	Korea
0x0413,	Dutch,	Netherlands
0x0813,	Belgian Dutch,	Belgium
0x0414,	Norwegian - Bokmal,	Norway
0x0814,	Norwegian - Nynorsk,	Norway
0x0415,	Polish,	Poland
0x0416,	Brazilian Portuguese,	Brazil
0x0816,	Portuguese,	Portugal
0x0417,	Rhaeto-Romanic,	Switzerland
0x0418,	Romanian,	Romania
0x0419,	Russian,	U.S.S.R
0x041a,	Croato-Serbian (Lat),	Yugoslavia
0x081a,	Serbo-Croatian (Cyr),	Yugoslavia
0x041b,	Slovakian,	Czechoslovakia
0x041c,	Albanian,	Albania
0x041d,	Swedish,	Sweden
0x041e,	Thai,	Thailand
0x041f,	Turkish,	Turkey
0x0420,	Urdu,	Pakistan
0x0421,	Bahasa,	Indonesia

os_type

Detecting the type of the operating system

int **os_type()**

Return The type of the operating system, can be one of the following:

- 2 DOS box under Windows NT.
- 1 DOS system.
- 0 Window 3.1
- 1 Window 95 and up
- 2 Window NT
- else** Reserved.

This function is also supported in DOS version of latest update.

outp

Output a single byte through a hardware port.

void **outp** (*port,data*)

int *port* I/O port address.

int *data* Data to be output. Only the lower byte is output by "OUT port,AL" instruction.

Note that this function will not work on NT machine.

outpw

Output a word through a hardware port.

void **outpw** (*port,data*)

int *port* I/O port address.

int *data* Word data to be output by "OUT port,AX" instruction.

Note that this function will not work on NT machine.

p or P

Intrinsic function to return a point by geometry definition.

POINT P (*argument...*)

Var *argument* Variable number of arguments of predefined type. Different number of arguments of different type will specify different definition of a point.

Return Point of the definition results.

There are many ways to define a point. To provide each of them with separate functions is cumbersome in these functions's naming as well as in referencing them. Recalling that the operator may directly enter a valid TCL expression to the command line and utilize the power of TCL expression to facilitate his drafting operation, lengthy naming would certainly be unacceptable. This is one of the major reason to provide a general point geometry definition function, **P()** or **p()**.

The methods used to define a point depends on the arguments passed to the function, and are described below:

- x,y** **Define a point by directly giving the X and Y coordinates.** This is the most frequently used method. X and y are double values. Integers will be converted to doubles automatically.
- rad,(ang)A** **Define a point by polar coordinates,** where rad is the radius value and ang is the angle value in units of degree. Note that the notation **()A** means the value must be tagged with an 'A', to notify it is an angle value. For example, *p(10,30A)* defines a point at 10 units away and in 30° direction from the origin. Should the use of tag character cause

any confusion, parenthesis should be used to make it clear, as in the example, $p(dist,(dir)A)$, where *dist* and *dir* are names of variable.

This is not a portable syntax to C/C++ language, but is convenient for operators in command line input. For portable reason, use the macro below:

```
#define polar(r,a) P(r,(a)A)
```

- ent** **Define a point by taken it from the given entity.** Valid argument types are CIRCLE, ARC, POINT, and ENTITY. For a circle or an arc, it is the center point taken. For a point argument, it is simply copied. If it is an entity handle, the drawing data pointed by it is read first and then checked for these valid entities.

- ent,length** **Define the point by measuring along a given entity** (the first argument) **at a given length or distance** (the second argument, double). Valid argument types for the first argument are LINE, ARC and ENTITY. For example, to locate a point in the direction from P1 to P2, at a distance of 10 units from P1, use the function call: $P(L(P1,P2),10)$, where the P1 and P2 are used first to construct the line via **L()** function call.

- ent,(ang)A** **Define a point by measurement based on a given entity** (the first argument) **at a given angle** (the second argument, double, tagged with 'A'). Valid argument types for the first argument are CIRCLE, ARC and ENTITY. The angle is in units of degree (must be tagged with 'A'). For a circle, the starting angle is always at absolute zero degree, while for an arc, it is the same as the arc's starting angle. If the first argument is an entity handle, the drawing data it points to will be read in and checked for the measurement. An example to obtain a point on a circle at 30° location is, $P(c1,30A)$, where c1 is the given circle.

- pnt,ent** **Define a point as the one mathematically on a given entity** (the second argument) **with the shortest distance to a given point** (the first argument of type POINT). Valid types of the second argument are LINE, ARC, CIRCLE, and ENTITY. If the argument is an entity handle, it will be loaded and checked for valid types. If the given entity is a line, this will return the base point on the line from the given point (i.e., the line from the given point to the base point will be perpendicular to the given line). If the given entity is an arc or a circle, a point on it (mathematically speaking) nearest to the given point is returned.

Note that this intrinsic function will be invoked automatically by the pure assignment operator (=) when the data type of the L-value is POINT. For example, the expression

```
P1 = 10,20
```

will be equivalent to the expression

```
P1 = P(10,20)
```

where the **P()** function is called by the assignment operator to evaluate the comma expressions to the right of it.

p_pctan

Find the tangent points on a circle such that the lines passing through a given point will be tangent to the circle at these tangent points, respectively.

int p_pctan (*p1,c2,result*)

- POINT *p1*** The given point for the tangent lines.
- CIRCLE *c2*** The given circle on which the tangent points lie.
- POINT *result[]*** Array of points to hold the returned answers.
- or
- LINE **result*** Pointer to a line to hold the returned answers.
- Return** Integer, number of tangent points found.

If the given point is outside of the circle, there are 2 such answers. If it is on the circle, itself is the trivial answer. If it is inside of the circle, there will be no answer at all.

pe_angle

Calculate the direction angle from a point to an entity.

double pe_angle (*pnt,ent*)

- POINT *pnt*** Given point.
- Var *ent*** Entity such as POINT, LINE, ARC, CIRCLE, and Entity handle pointing to Line, Arc, Circle and Polyline.
- Return** Direction angle of the shortest line from the given point to the given *ent*, in units of degree. See also **pe_dist()**.

This function is supported only after V3.1.055.

pe_dist

Calculate the distance from a point to an entity.

double pe_dist (*pnt,ent*)

- POINT *pnt*** Given point.
- Var *ent*** Entity such as POINT, LINE, ARC, CIRCLE, and Entity handle pointing to Line, Arc, Circle and Polyline.
- Return** Distance from the given point to the given entity. If the entity is a POLYLINE, it return the possible shortest distance from the point to the polyline. For all other entities, it returns the mathematical distance from the geometry entity.

If the object is a polyline, positive distance means the point is to the right of the polyline, and negative means to the left. The direction of a closed polyline can be determined by the sign of the calculated area. CCW for positive area, CW for negative area. The **pe_dist()** is thus can be used to determine whether a point is inside of a polyline or outside of it.

This function has been provided in **v()** or **V()** function calls, but most of the programmers may not have noticed this fact.

PenColor Win

Get/Set physical pen color of drawing space.

long PenColor (*which, color*)

int *which* Physical pen color number, currently ranged from 0 to 15, and will be expended to 256 in future.

long *color* Optional RGB color to re-define the specified physical pen color. If this argument is not given, the specified pen color will not be re-defined.

Return The RGB color of the specified physical pen. If the physical pen is to be re-defined, this gives the last RGB color of the pen.

The physical pen color is the palette color used by TwinCAD to realize the drawing's logical pen. Currently, only 16 palette colors are supported. TwinCAD uses the physical palette color index 0 as the drawing background on the screen. So, change palette color index 0, will effectively change the drawing's background (need regeneration to see the change).

pi

Intrinsic function to return PI value of maximum precision.

double pi()

Return 3.141592653589793238462643383279...

You may directly reference the **pi()** function as a data variable as **pi**.

plotent

Plot basic entities, such as points, lines, arcs and circles, to the current viewport without actually creating them.

void plotent (*entlist...*)

entlist

- char str[]** Text string to be plotted using current setting of text style and default text control informations.
- POINT pins** Specifying where to plot the text string. This is equivalent to the text insert point.
- double angle** Optional text angle in units of degree, default to 0.

This extended function call is used to generate text in the drawing area without actually creating them to the drawing database.

pm_command Win

Issue Program Manager Command

int pm_command (cmd,...)

char cmd[] Program Manager DDE command strings. Multiple commands can be given in successive string arguments.

Return TRUE, if the DDE connection with the Program Manager is successful and the command string has been passed; otherwise, it returns false.

This function is used to command the Windows Program Manager for Program Group and Items manipulation. It is a part of the **TPInstall**'s runtime function specification. **TPInstall** is an independent TCL interpreter designed for general program installation under Windows environment.

An example call of **pm_command()** to create program group for TwinCAD is given below:

```
pm_command("[CreateGroup(TwinCAD V3.1,TwinCAD)]",  
          "[AddItem(C:\\TWINCAD\\TWINCAD.EXE,TwinCAD V3.1)]");
```

Note that the command string must be enclosed in a pair of bracket. See appropriate documents from Windows for these **PM** commands.

pm_request Win

Request Information from Program Manager

int pm_request (item, data)

char item[] String of the item data to request.

char data[][] String array to hold the returned data information.

Return Number of strings returned by the **PM**.

This function call is used to retrieve informations from the Program Manager for program groups and group contents. To request the listing of program groups, call this function with item name "Groups" as in the example given below:

```
int i, nGroup;  
char Groups[50][64];  
nGroup = pm_request("Groups",Groups);  
for(i=0;i<nGroup;i++)
```

```
printf("\nGroup %d: '%s' ", i, Groups[i]);
```

Each string record returned will contain the name of a program group. Note that the appearing orders of these group names are in the same order they are appeared in the PM's main window, from left to right and from top to bottom.

To obtain the details information about a specific program group, call this function with the group name as the item name in request, as the example given below:

```
char Items[50][256];
nItem = pm_request("TwinCAD V3.1", Items);
```

The fields of group information in the strings are separated by commas. The first string of the information contains the group name (in quotation marks), the path of the group file, and the number of items in the group. Each subsequent string contains information about an item in the group, including the command line (in quotation marks), the default directory, the icon path, the position in the group, the icon index, the shortcut key (in numeric form), and the minimize flag. The TCL application can use **sscanf()** to retrieve each part of these information from the returned string.

See appropriate documents from Windows for more details.

polyeval

Evaluate a polynomial of given order.

double **polyeval** (*nord,coef,tval*)

or

POINT **polyeval** (*nord,pcoef,tval*)

int *nord* Order of polynomial

double *coef[]* Coefficient of polynomial, *nord*+1 elements, for a one-dimensional polynomial function:

$$F(t) = C_0 * t^n + C_1 * t^{n-1} + \dots + C_{(n-1)} * t + C_n$$

Where $C_0 \dots C_n$ are given by *coef[0] ... coef[n]* respectively.

POINT *pcoef[]* Coefficient of polynomial, *nord*+1 elements, for a two-dimensional parametric function:

$$F(t) = B_0 * t^n + B_1 * t^{n-1} + \dots + B_{(n-1)} * t + B_n$$

where $B_0 \dots B_n$ are given by *pcoef[0]...pcoef[n]*, respectively.

double *tval* Real parameter value to evaluate the function.

or

POINT *tval* Complex parameter value to evaluate the function. The parametric function is taken as a polynomial with coefficients in complex numbers. The evaluated result is always expressed in complex number (POINT).

Return Double or POINT value of the evaluated result.

The maximum order of polynomial is limited to 40 at current version.

polyflag

Set special flag to polyline.

ENTITY polyflag (*ent,flag*)

Entity *ent* Entity handle to a polyline of which the entity flag is to set.

int *flag* Flag value to set to polyline entity. Current valid bit flags are **TOOLPATH** (4) and **OPENPATH** (8). All other bit flags are ignored.

This function is intended for use with TCL programs that produce tool paths. A tool path is a polyline with a **TOOLPATH** bit flag set at the entity flag, and will be treated specially in **TwinCAD**. Note that the linetype generation of a tool path will be disabled.

PostMessage Win

Post message to given window.

long PostMessage(*hwnd, msg, wParm, lParm*)

int *hwnd* Window handle (not field ID).

int *msg* Message code to post. See Windows SDK document for useful message code.

int *wParm* Word parameter to post to the window.

long *lParm* Long parameter to post to the window.

Return -1, if the window handle is not valid. 0, if the operation fails.

This function directly calls Windows API PostMessage() to post specific message to specific window. Care should be taken in using this function. **TwinCAD** can not prevent this function call from resulting any serious problem. Those who use this function are assumed to be well understanding about the Windows messages. See Windows SDK document for more informations.

Note that there is no way for the time being to pass a data pointer to receive data returned from the called window. Trying to do so, would likely cause GP error. This function is hidden from the **CMDLIST**.

Use **wnd_handle()** to obtain the window handle of specific field ID.

printf

Formats and prints a series of characters and values to the standard output device. Unless redirected, the standard output device will be the command text area in the display.

void printf (*format[,argument]...*)

char *format[]* Format-control string.

Var argument Optional arguments of any type as specified in the format-control string.

The control string is simply a text string to be copied to the output, except that the string may contain conversion specifications. A conversion specification may require to process some additional arguments and result in a formatted conversion operation that generate the output characters not explicitly contained in the control string. For each conversion specification in the control string, there should be a corresponding argument of exactly the right type; otherwise, the results are unpredictable. Also, if any conversion specification is malformed, then the effects are unpredictable.

A conversion specification begins with a percent sign '%'. This make the output of a percent sign '%' requiring two consecutive '%'s in the control string. Following the percent sign '%', the following conversion specification elements should appear in the order below:

- **Zero or more optional flag characters,**
 - '-' (**minus**) Specifying to justify the converted string to the left of the field within the field width if applicable, rather than the default right-justification.
 - '0' (**zero**) Specifying to use the '0' for the pad character rather than the space.
 - '^' (**caret**) Specifying to justify the converted string to the center of the field within the field width if applicable and possible, rather than the default right justification.
 - ',' (**comma**) Specifying to punctuate the decimal string output with commas, such as "1,000,000".
- **An optional field width,** expressed as:
 - ⇒ a decimal integer constant (sequence of decimal digits), or
 - ⇒ an asterisk '*', which takes an integer argument to specify the field width. This will consume one argument from the argument list.
- **An optional precision specification,** expressed as a period '.' followed by either:
 - ⇒ an optional decimal integer, or
 - ⇒ an asterisk '*', which takes an integer argument to specify the precision. This will consume one argument from the argument list.
- **An optional long size specification,** expressed as the character 'l' (lowercase letter L), which is used in conjunction with the conversion operations 'd', 'o', 'u', and 'x' to indicate that the argument is long.
- **A required conversion operation,** expressed as a single characters:
 - c** The argument is output as a single ASCII character.
 - s** The argument is output as a text string.
 - b** The argument is taken as a signed integer or long (if 'l' is used) and converted to its binary string equivalence for output.
 - B** The argument is taken as a signed long and converted to its binary string equivalence for output.

- d** The argument is taken as a signed integer or long (if 'l' is used) and converted to its decimal string equivalence for output.
- D** The argument is taken as a signed long and converted to its decimal string equivalence for output.
- o** The argument is taken as a signed integer or long (if 'l' is used) and converted to its octal string equivalence for output.
- O** The argument is taken as a signed long and converted to its octal string equivalence for output.
- x** The argument is taken as a signed integer or long (if 'l' is used) and converted to its hexadecimal string equivalence for output.
- X** The argument is taken as a signed long and converted to its hexadecimal string equivalence for output.
- u** The argument is taken as an unsigned integer or unsigned long (if 'l' is used) and converted to its decimal equivalence for output.
- U** The argument is taken as an unsigned long and converted to its decimal equivalence for output.
- e** The argument is taken as a floating point value and is converted into its scientific representation as in the form: "*[-]d.dddddE+dd*". The precision specifies the number of digits to be printed after the decimal point.
- f** The argument is taken as a floating point value and is converted into the decimal string equivalence as in the form: "*[-]ddd.dddd*". The precision specifies the number of digits to be printed after the decimal point. Note that if the conversion output should overflow the field, the 'e' conversion output will be taken instead.
- g** The argument is taken as a floating point value and is converted according to the value of the argument. If the argument is too large or too small, the e format will be used; otherwise, the f format is used. Note that if necessary, the precision number may be adjusted automatically so that the output field may present the value in maximum precision as possible.

The use of the conversion specifications in the control string must be very careful. If the rules are not followed correctly, the output result may be unpredictable.

prints

Formats and prints a series of characters and values to a string.

STRING **prints** (*format[,argument]...*)

char *format*[] Format-control string.

Var *argument* Optional arguments of any type as specified in the format-control string.

Return The resulting string of output.

Note that this function implements the same function as the **sprintf()** in standard C library, however, in different form. The **sprintf()** function can be implemented as a macro:

```
#define sprintf(s,fmt,...) s=prints(fmt,...)
```

See **printf()** for a discussion of the formatting operation.

put_map

Draw previous defined map at specific position.

void **put_map** (*id, pbase*)

or

void **put_map** (*id, xbase, ybase*)

int *id* ID number of the map, ranged from 0 to 15.

POINT *pbase* Where to draw the map.

double *xbase* X-coordinate of the position to draw the map.

double *ybase* Y-coordinate of the position to draw the map.

This function will directly call the **TCAM/Graphic Runtime** to draw the previous defined drawing map of specific ID number at the specific location in the drawing window. The map will be drawn under current setting of scaling and rotation setup, set by the **set_map()** function calls. The color of the map is controlled by the **setplot()** function.

If the drawing map was not defined or already invalid, then nothing will happen. No error condition will be reported or signaled.

See also **set_map()** for detail informations about how to define a drawing map.

put_profile Win

Write parameter string to specific profile

int **put_profile** (*section, entry, pstr, inifile*)

char *section* The parameter section name.

char *entry[]* The parameter entry name.

char *pstr* String value to put to the profile.

char *inifile[]* Optional name of the profile file. If this argument is not given, it defaults to TwinCAD's current active profile file. If it is given without path, it defaults to the Windows directory.

Return TRUE, if successful, FALSE if not.

The parameter string in the profile will be in the form as given below:

```
[Section]
Entry = String
....
```

See also **get_profile**.

pwr

Intrinsic function to implement the '**' (power) operation.

Var pwr (args...)

Var args Variable number of arguments of applicable data type. The minimum number of arguments are 2, while the maximum number are 15. Type conversions are performed automatically from left to right in sequence.

Return Result of the power function, depending on the type of arguments.

This is the intrinsic function to implement the '**' (power) operation in the expression. In other words, the expression: $a**b**c**...$ can be written as $PWR(a,b,c,...)$. Note that both upper case and low case of the function names are accepted. The type of operands can be integers or doubles. Type conversions from integer value to double are performed automatically when they are mixed in the operation. table below:

Operand1	-	integer	double	point
Operand2		-----		
integer		integer	double	point
double		double	double	point
point		point	point	point

The actual implementation of $PWR(a,b,c,...)$ is $PWR(a,MUL(b,c,...))$. This function implements the **pow()** function from the standard C library. However, this function accepts integer arguments and may produce integer result.

Special Notes on pwr()

The call to **pwr(val,0.5)** is not exactly equivalent to the call to **sqrt(val)** in that the **pwr()** accepts negative value in its first argument and it will produce negative result for the case. However, the TCL programmer should use **sqrt()** instead of **pwr()** if possible, since the implementation of **sqrt()** in TCAM system can produce the maximum precision of the result even without the Math-Coprocessor, and is much faster than **pwr()**.

For example, the following result is produced with the use of Math-Coprocessor:

```
@CMD: printf("\n%16.14f",sqrt(2))
1.41421356237309
@CMD: printf("\n%16.14f",pwr(2,0.5))
1.41421356237309
```

Without the Math-Coprocessor, the result will be:

```
@CMD: printf("\n%16.14f",sqrt(2))
1.41421356237309
@CMD: printf("\n%16.14f",pwr(2,0.5))
1.41421356237293
```

Apparently, the error produced by **pwr()** using software implementation in 64-bit precision will propagate to the 13 digits, recalling that the maximum precision is 14.5 digits. The exact **sqrt(2)** value in 20 digit precision is 1.4142135623730950488, given here as a reference for comparison.

The definition to **pwr()** in TCL library function is given below:

```
pwr(x,y) :=  
0          if x is zero  
exp(y*ln(x))  if x is positive  
exp(y*ln(-x)) if x is negative and y is even integer  
-exp(y*ln(-x)) if x is negative and y is odd or not integer
```

Obviously, writing X^*X^*X is better than writing **pwr(X,3)**.

The implementation of **pwr()** is slightly different from the standard C function **pow()** which returns 0 if x is negative and y is not integer.

r_level

Determine the containing relationship among given close entities.

int **r_level** (*slist,level*)

SLIST *slist* Selection list of non-intersecting close polylines and circles.

int *level[]* Array of integer to hold the returned level indicators; each element corresponds to each entity from **slist**. A -1 means un-determined, 0 means the top level profile, 1, holes in the top level profile, 2, islands in the holes, etc.

Return Number of effective close objects found from the **slist**.

r_simplex

Check/Modify the connectivity of a close region bounded by a polyline or other region primitives.

int **r_simplex** (*ent,chktype,parray,se1,se2*)

ENTITY *ent* Entity handle pointing to close polyline or region primitives.

int *chktype* Optional checking option, default to 0. The meaning is given below:

-1 Do check and simplify, return number of additional simplex regions produced, and separation points in *parray*, and adjacency sets in *se1* and *se2*.

0 Do check and simplify, return number of additional simplex regions produced.

1 Do check only, return number of self-intersection points.

2 Do quick check only, return 1 if *ent* has self-intersection point.

3 Do check only, return number of intersection points, and points in *parray*.

Else Reserved, Treated as 0.

POINT *parray[]* Optional, point array to hold the returned intersection points.

SLIST *se1[]* Optional, Selection list to hold the first set of adjacency relationship.

SLIST se2[] Optional, Selection list to hold the second set of adjacency relationship.

Return Depending on the operation type, as described below:

- 1** The given entity is not a valid region primitive.
- 0** The given entity is a valid simplex region primitive already.
- N>=1** If *chktype* is 0 or -1, it means the given entity is not a simplex region primitive but can be separated to N+1 simplex region primitives. The original entity will be modified to a simplex one, and additional N entities will be produced.

If *chktype* is 1 or 3, it returns the number of self-intersection points.

If *chktype* is 2, it always returns 1 if there is a self-intersection point found.

If *chktype* is -1, this function will expect two entities selection lists and a data point array passed in the argument list. If N+1 simplex regions were separated, then there will be N points at which these regions separated. It returns the N data points in the passed data point array, and in corresponding to each data point, the entity handles of the two separate simplex regions are stored in the passed selection list respectively. That is, entity *se1[0]* and *se2[0]* will be adjacent by the data point *parray[0]*.

It is also guaranteed that there is no duplicated handle in the first selection list, so that the first selection list is also the selection list of those newly created simplex regions (excluding the original one).

This is an advanced TCL function. The use of this function is intended for specific applications and may be restricted to those separately authorized users in the future.

Special Notes on Region Primitives

Current valid region primitives that can be passed to this function is closed POLYLINE only. This function will check if the polyline has twisted by itself. It will separate the polyline into several close polylines (simplex regions) by the points where it twists to intersect itself.

All the polyline segment's direction are preserved in the newly created polylines. So, if the given region is only simple twisted, then from the sign of the area of these resulting closed polylines, it can be very easy to tell which one is reversing the direction of area vector (the one twisted from the original polyline).

A simple twisted region is defined as one that can be produced by twisting a rubber band without folding, and the twisted part does not overlap with the main part. A complicated twisted region is defined as one that can not be produced by twisting a rubber band without folding or without overlapping. A simplex region produced by twisting it from a rubber band (the main region) will have a reversed area direction vector. The folding will reverse it again. Folding will

undone by **UNDO** command. However, If more than two simplex regions are resulted from the operation, once it is **UNDO**ne, it can not be **REDO**ne.

rand

Intrinsic function to return a pseudo-random number.

int rand()

Return Integer pseudo-random number ranged from 0 to 32767.

rdms

Read/Convert DMS-spec angle string into equivalent degree value.

double rdms (str)

char str[] String, containing angle value in DMS-spec format (Degree-Minutes-Seconds).

Return Equivalent degree value in floating point.

This function is useful for data input at command line where the operator may wish to input an angle value in DMS-spec format.

The general format of a DMS specification is given below:

{-}#[d]D|.|院:|#[']|.#{...}

where # represent decimal values. Valid examples are as below:

-10:15.30.5
-10d15'30.5"
-10°15'30.5"
-10.15.30.5

Note that this is not an intrinsic function, and so it must be typed in lower case.

redraw

Direct call to **REDRAW** command or refresh window.

int redraw(int wid)

int wid Optional window handle as returned by **wndopen()** or **wndopensub()**. If this argument is given and is valid, it causes the specified window repainted. If this argument is not given, it redraws the drawing area. In DOS version, the function call always redraws the drawing area, in despite of the presence of this argument.

Return In Windows version, if the optional argument is given, it returns the same argument value if the argument is a valid window handle or returns 0 if it is not.

redraw_count Win

Check to see if the drawing area had been refreshed since the last checking.

int redraw_count()

Return The number of drawing refresh has been done since the last call to this function.

This functions is useful in detecting whether the operator has made the drawing area refreshed during the user interface.

regen

Direct call to **REGEN** command.

void regen()

These above two functions can be achieved by **command("^R ")** and **command("^N ")**. However, using the two newly added functions will make the program more readable, and faster in response.

regen_count Win

Check to see if the drawing had been regenerated since the last checking.

int regen_count()

Return The number of drawing regeneration has been done since the last call to this function.

This functions is useful in detecting whether the operator has changed the drawing window during the user interface.

remove

Erase specific file.

int remove (*fname*)

char *fname*[] Name of the file to erase.

Return 0 if the operation is successful, else the error code returned from DOS.

- 2** File not found.
- 3** Path not found.
- 5** Access Denied.

This is an ANSI standard function. The Unix equivalent **unlink()** can be implemented via **#define**.

This function directly makes call to DOS, and is much faster than using `command("DOS DEL filename")`. However, this function erases only one file once in a time. No wildcat characters can be used in the file-spec, as that can be done via the DOS command.

rename

Rename a specific file from one name to another.

int **rename** (*oldname,newname*)

char oldname[] Name of the file to rename.

char newname[] New file name to rename to.

Return 0 if the operation is successful, else the error code returned from DOS.

- 2** File not found.
- 3** Path not found.
- 5** Access Denied.
- 0x11** Not the same device .

This function directly makes call to DOS, and is much faster than using `command("DOS REN oldname newname")`. However, this function renames only one file once in a time. No wildcat characters can be used in the file-spec, as that can be done via the DOS command.

rewind

Move the file position counter of a file stream to the beginning of file.

void **rewind** (*fid*)

FILE fid Stream ID (integer in actual implementation).

This function can be implemented as `fseek(fid,0L,0)`.

right\$

Intrinsic function to take a part of a string from its end.

STRING **right\$** (*str,n*)

char str[] Source string.

int n Number of characters from the end of the string to take.

Return Sub-string taken from the first argument up to the specific number of characters from its end. If *n* is larger than the length of the source string, then the whole string is returned.

rmdir

Remove directory with a specified name.

int rmdir (*dirname*)

char *dirname*[] Pathname of the directory to remove.

Return 0 if the operation is successful, else the DOS error code:

- 3** Path not found.
- 5** Access denied (if the directory to be deleted is not empty or the directory to be deleted is the root directory).
- 0x10** Current directory error (you can not remove current directory).

rnd

Intrinsic function to round the argument to a nearest exact mathematical integer

double rnd (*value*)

double *value* Double value to round.

Return Double value that is a nearest exact mathematical integer to the argument.

rnd2

Intrinsic function to round the first argument to a nearest multiple of the second argument.

double rnd2 (*value,unit*)

double *value* Double value to round.

double *unit* Unit value for the rounding.

Return Double value that is a nearest multiple of the unit value.

The **rnd()** function can be implemented by this function as **rnd2(value,1.)**, as the rounding unit is one.

root2

Find the real roots of a polynomial of order 2

int root2 (*a,b,c,ans*)

double *a,b,c* Coefficient of polynomial $A*X^2+B*X+C=0$.

double *ans*[] Array to hold the returned answers.

Return Number of real roots found:

- 0** No real solutions.
- 1** One redundant solution.
- 2** Two real solutions.

root3

Find the roots of a polynomial of order 3

int **root3** (*a,b,c,d,ans*)

double a,b,c,d Coefficient of polynomial $A*X^3+B*X^2+C*X+D=0$

double ans[] Array to hold the returned answers.

Return Number of real roots found:

- 0** No real solutions. (impossible!)
- 1** One real solution, and two real complex roots (R1, P, Q)
- 2** Two real solutions, one coincides (R1, P, 0).
- 3** Three real solutions (R1, R2, R3)

A polynomial of odd degrees must have at least one real root. So, it is not possible for **root3()** to return 0. A cubic polynomial always has three roots. One is always real, and the other two are either also real or conjugate complex. Let the reals be R1, R2, R3, and conjugate complex be P+/-Q, where P is the real part, and Q the imaginary part, then this function will return the roots in the *ans[]* array as parenthesized above.

rtd

Intrinsic function to convert angle value from units of radians to units of degree.

double **rtd** (*angle*)

double angle Angle value in radians.

Return Equivalent angle value in units of degree.

run

Execute an external TCL program file.

int **run** (*pgmfile, tpakey*)

char pgmfile[] Name of the TCL program file to run. The file can be in TCZ or TCA format.

char tpakey[] Optional authorization key of registered third party. This key is checked only under Lite Mode (Base Mode) operation. Without a valid activation key, **run()** can not execute an external TCL/TCZ program file under Lite Mode (Base Mode).

Return 0, if successful.

This function call is equivalent to the expression `command("RUN pgmfile")`, but with less overhead.

sarc

Produce spline arcs for parametric curves approximation.

int **sarc** (*ps,pds,pe,pde,arcs*)

POINT *ps* Position Vector of start point.

POINT *pds* Tangent Vector at start point.

POINT *pe* Position Vector of end point.

POINT *pde* Tangent Vector at end point.

ARC *arcs[]* Array of arcs to hold the result.

Return Number of arcs resulted from the given vectors. At most 2 arcs are used for the approximation. There may be only one arc for special cases, or no arcs at all if the vector change dramatically.

Note that the Dual-arc approximation assumes that the parametric curve is very smooth and the change of radius of curvature is reasonably 'small' and is distributed evenly in the given interval. There should be no inflection point in the interval.

For a given 2-D parametric functions:

$$\mathbf{X} = \mathbf{X} (t)$$

$$\mathbf{Y} = \mathbf{Y} (t)$$

where the parameter t varies over a given range $t1 \leq t \leq t2$. The position vector at $t0$ is directly obtained by $P(x(t0),y(t0))$, and the tangent vector at $t0$ is then by $P'(x(t0),y(t0))$, which is $P(x'(t0),y'(t0))$.

The detail generation of Dual-arc approximation is controlled by the system variable **@SARCTYPE**. See also **@SARCTYPE** in part 2 of this document.

search

Search for a key value in an array, sorted or unsorted.

int **search** (*key,base,nemt,flag*)

Var *key* The key value of various type to search for from the given array. Supported data types are integer, long, double, POINT, string and ENTITY.

Var *base[]* Array of values to search from.

int *nemt* Optional number of element in the array. If this is not given, the whole array will be searched through.

int *flag* Optional, default 0, flag value to specify the search options, as described below:

- Bit 0:** 1, if the array is sorted in descending order, or 0, if in ascending order, valid only when bit 7 is ON.
- Bit 1:** 1, if the array is sorted by Y coordinate first, or 0, if it is by X coordinate, valid only when the data type of the array is of POINT type and the bit 7 is ON.
- Bit 7:** 1, if the given array is in sorted order, 0 if it is not.
- else** Reserved and should be zero.

Return -1, if the key value is not found from the array, or else, the index of the key from the array.

Data type conversion will be done automatically to the key value if its type is not the same as the given array. However, if the conversion is not possible, a data type error will occur.

If the given array is in sorted order, the **search()** function will perform binary search for the key; otherwise, linear search will be done.

sel_icon

Pop up an icon menu and get user response.

int sel_icon (title, iconspec, rowno, colno, dist)

char title[] Title string for the icon menu. If the passed argument is not of character string type, or is a null string, then the pop up icon menu will not have a title bar; otherwise, a title bar will be given with the title string in it.

char iconspec[][] Array of character strings containing the icon specifications. The number of icon is determined by the number of strings passed by the array; however, a null string in the array will terminate the counting of the icon number.

int rowno Optional row counts of tilts. See description.

int colno Optional column counts of tilts. See description.

int dist Optional distance factor between tilts.

Return The order number from 0 to n, where n is the number of icon minus one, indicating which icon from the menu is picked up by the operator. Should the operator press <ESC> or <Ctrl/C>, or pick up the [-] button of the menu window if a title bar is present, this function will return a negative value of -1.

The specification of an icon in a string can be

- The filename of a slide file without any leading spaces, or
- The filename of a slide library without any leading spaces, followed by the name of a slide in the library enclosed with a pair of parenthesis, or
- A text string with the first character being a space, to be display as text in the icon box.

Such specifications are the same as those for the Icon Menus specified in the menu file, so is the rule for the display of icons.

The maximum number of icons can be displayed at the same time is limited to 81. However, further limits will be set by the **rowno** and **colno** that specifies the number of tilts in the display.

The maximum number allowed for each **rowno** and **colno** is 16. If one is not given or less than zero, a zero value will be assumed. They are used to determined the arrangement and total number of tilts created in the window to hold the icons display.

If both are not zero, then a **colno x rowno** (X x Y) array of tilts will be created to hold the icons. If either one is zero, the one will be calculated by dividing the total number of icons by the other one, and rounded to the next integer.

If both are zero (as by default if both are not given), they will be calculated to have same value to hold all the icons (such as 2x2, 3x3, ...9x9).

The distance between tilts can be controlled by the **dist** factor in units of about 6 to 8 dots. If the **dist** is not given, it will be default to 2. Note that the frame of the tilts will narrow the distance between them.

Though the number of icon specifications is determined by the number of string (a single array of character) elements in the passed array, it is better to include a null string at the end of the string array to avoid possible porting problem in the future. Note that in standard C implementation, the argument will be passed as a pointer to an array of pointer to character string (***icon[]**), and there may be no way for the function to know

long IParm Long parameter to send to the window. String type parameter is also accepted as a string constant and will be converted to far data pointer to pass for readonly purpose.

Return Long value returned by the window.

set_drag

Setup dragging requirement for the next **getinput()** function call.

void set_drag (cmd, pbase, angle, rlen, slist)

- int cmd** Type of the dragging requirement, see description later. If -1, the dragging will be disabled; otherwise, the dragging is enabled.
- POINT pbase** Base point for the dragging.
- double angle** Optional reference angle for the dragging.
- double rlen** Optional reference length for the dragging.
- SLIST slist** Optional selection list for the dragging.

This function call will setup the required parameters for the system's internal dragging handler to work for the subsequent **getinput()** function calls. It must be called immediately before **getinput()** function.

The first argument *cmd* is used to specify the type of the image dragging, as described in the table below:

- Bit 0-3:** Basic Dragging type, as listed below:
- 0** RotOFF, put_map(0) at CP.
 - 1** RotOFF, put_map(0) at CP, dashline(CP,pbase).
 - 2** RotByCP, put_map(0) at pbase, dashline(CP,pbase).
 - 3** RotByCP/Ref, put_map(0) at pbase, and dashline(CP,pbase).
 - 4** RotOFF, put_map(0) at (pbase.x,CP.y), put_map(1) at CP, line((pbase.x,CP.y),pbase), dashline(CP,pbase).
 - 5** RotOFF, put_map(0) at (CP.x,pbase.y), put_map(1) at CP, line((CP.x,pbase.y),pbase), dashline(CP,pbase).
 - 6, 7** Internal use. Don't use in TCL Programming.
 - 8** Nothing
 - 9** RotByCP/Ref, ScaleByRef, put_map(0) at pbase, and dashline(CP,pbase).
 - 10-14** Reserved, Same as 0.
 - 15** RotOff, put_map(0) at pbase, dashline(CP,pbase)
- Bit 4:** On, use solid_line() instead of dashline().
- Bit 5:** On, if crosshair is disabled, drag a big cross (+) mark at CP.
- Bit 6:** On, disable crosshair (also disable by bit 7, 0x80).
- Bit 7:** On, Disable the dashline(CP,pbase).
- Bit 13:** ON, create drawing map on selection list, the *slist* must be given. OFF, if no new drawing map to create. The TCL application must create its own

drawing map using `set_map()` before calling `getinput()`, if a dragging map is to use.

Else Reserved, should be zero.

The notations used in the above table are described below:

CP	The dynamic Cursor Position.
pbase	The pre-setup base point given in the argument.
RotOFF	Turn-off rotation while putting the drawing map.
RotbyCP	Rotate the drawing map around the base point by the cursor position.
RotbyCP/Ref	Rotate the drawing map around the base point by the cursor position with respect to a reference angle given in the argument.
ScaleCP/Ref	Scale the drawing map by the cursor position about the base point with respect to a reference length given in the argument.
dashline(p1,p2)	Draw a dash line from p1 to p2.
line(p1,p2)	Draw a solid line from p1 to p2.
put_map(n)	Put the drawing map #n to the screen. See also put_map() .

Note: If a dragging drawing map is not required for the next **getinput()** function call, use **set_map()** function to explicitly clear the drawing map definition from the Graphic Runtime. This will avoid the dragging of an undesired drawing map resulted from the last user interface operation (such as that from command("COPY...")). The **set_drag(-1)** will not clear the drawing map definition automatically. However, it is recommended to issue a **set_drag(-1)** after the **getinput()** call, since the parameters setup by **set_drag()** will remain effective until it is reset.

set_ecs

Setup ECS plane data for subsequent new entities in creation

int set_ecs (dirx,diry,dirz,auxzh)

double dirx	X-component of ECS plane Normal Vector
double diry	Y-component of ECS plane Normal Vector
double dirz	Z-component of ECS plane Normal Vector
double auxzh	Optional, Entity Elevation in ECS plane for Text/Dimension.

Return 1, if ECS effective now, 0, if ECS is turned off (coincide with WCS).

The ECS transformation of planar data conforms to the AutoCAD's arbitrary axis algorithm. Additional functions are also provided to facilitate the transformation calculation.

The use of this function is provided for TCL to create 2-D objects in 3-D space. However, the TCL is responsible for all the coordinate system calculation. The TCL program must turn off the ECS transformation before exiting the program; otherwise, the user may be confused in subsequent operation.

To turn-off this ECS transformation, use **set_ecs(0,0,1)**. See also **ecs_org()**, **wto_ecs()** and **eto_wcs()** functions for related informations.

set_eps

Specify the epsilon value for double comparison.

double set_eps (eps)

double eps Optional epsilon value for double comparison, must be positive. If the argument is not a double value, it will be ignored.

Return Current epsilon value setting.

Testing two double values for exactly equal is sometimes impractical, due to the fact that the accumulation of calculation error (rounding/truncation) will break the equality of two theoretically equal values. For example, the value returned from **rtd(dtr(30.))** will probably have a slightly difference with the original value of **30**, since it is multiplied by **pi/180** and then by **180/pi**, while both of these two values can not be exactly represented as binary numbers (nor in decimal). You thus can not expect the expression $(\pi/180) \cdot (180/\pi)$ to return the exact value of 1 as you do by the math simplification.

It is therefore required to redefine the definition of equality between two values, as stated below:

Two value is said to be equal if and only if the difference between them lies within a small value of epsilon (i.e., less than and equal to the epsilon value in binary representation).

Mathematically speaking, this epsilon value may be an infinitesimal, while in practice, it must be a value representable in binary number.

Be aware that the setting of this value will remain effective until the **TwinCAD** exits. Care must be taken in setting this value. Setting a zero value will force to test for exact equality of two values in representation. A negative value will deny all equality tests. However, it affects only the result of expression evaluation, not the CAD system. The internal default setting when the **TwinCAD** is started up is **0.00000001**, which should be acceptable for most of the CAD applications. If you must change this value for your applications, restore it before your applications exit.

set_fkey DOS

Setup a specific function key in display and optionally its associated menu script content, and to provide some other related operations.

void set_fkey (cmd,parm,...)

int cmd Command word to specify the function request, see description.

Var parm... Optional parameters for the specific function request.

The **set_fkey()** function will directly call the **TCAM/Graphic Runtime** to manipulate the function key display and related setup operation. The following describes the details of the **set_fkey()** function under different command word:

Set function key display name and associated menu script

void set_fkey (id,name,script)

int id Function key ID number from 1 to 10, for F1 to F10 respectively.

char name[] The display name of the function key.

char script[] Optional menu script content to associate with the specific function key.

This function will put the given string (the argument name) to the specific function key in display. The function key F1 to F10 is identified by the ID number from 1 to 10 respectively. If the optional argument script is given, then the actual function key definition for the key in the menu system will be replaced by the new definition. If the script is not given, it affects only the function key in display temporary.

The maximum length for the function key name in display varies with the type of display. However, for standard VGA/EGA, it is 6 display characters. The graphic driver will automatically truncate the exceeded characters.

Set function key attribute

void set_fkey (-3,color,acolor,bcolor,type)

int color Color code for function key in normal display.

int acolor Color code for function key in Active display.

int bcolor Color code for function key in Activated display.

int type Type of function key display, 0 for full height and 1 for half height.

This function call changes the display attribute of the function keys in display. These arguments are corresponding to the initial setup in the INI file under the item: **"FuncKeyAttribute="**.

The acolor is used when the cursor pointer is crossing over a function key. The bcolor is used for the activated display of a function key.

Miscellaneous Functions

void set_fkey (id)

int id See below description.

0 Clear all the function key names in the display, and also disable the function keys.

- 1 - 10** Setup function key display, as described earlier.
- 11 - 20** Change the specific function key to activated color. The function key F1 to F10 corresponds id code from 11 to 20, respectively.
- 21 - 30** Change the specific function key to normal color. The function key F1 to F10 corresponds id code from 21 to 30, respectively.
- 1** Disable the activation of all function keys. The cursor pointer will no longer activate the function keys when it is crossing over them.
- 2** Enable the activation of all function keys. A function key that has been defined in the display will be activated when the cursor pointer is crossing over it and can be picked up to generate the corresponding function key code.
- 3** Set function key attribute, as described earlier.
- 5** Enable Floating Function keys. The function key display will show up only when the cursor pointer is crossing over. Note that it will not make function keys currently in display to disappear.
- 6** Disable Floating Function keys. The function key display will be fixed in the screen. Note that it will not make the function key to float up and stay upon being called. It affects the subsequent function key setup operation.
- 9** Restore the function key display attribute back to the system initial setup.
- Else** Ignored and reserved for future expansion.

The use of these above function calls should be careful, since they will disturb the screen display. The system will not make any effort to fool-prove a TCL programmer from doing so.

The system will automatically resume the function key attribute to system default and restore the function key definition from the menu setup whenever the last GUI window either opened by a Prime Command (such as **DMODE**) or by the TCL application is closed.

Note that it is not possible to restore the image covered by the function key display if it is not floating. However, since the command area may overlap with the function key display, scroll up the command window may clear it out.

set_map

Drawing Map Manipulation function

void set_map (cmd,parm,...)

int cmd Command word to specify the function request, see description.

Var parm... Optional parameters for the specific function request.

The **set_map()** function and the **put_map()** function together provide the TCL programmer a way to handle fast display image manipulation. Such image manipulations are extensively used by the system in object dragging. A drawing map is such an image created for the purpose. It is a set of display lists maintained by the **TCAM/Graphic Runtime** and referenced by an ID number from 0 to 15.

Once a drawing map is defined, the generation of the image to the screen will become very easy and fast to the programmer. The image can be rotated, scaled and drawn to any place the programmer desired simply by the **set_map()** and **put_map()** function calls.

The following describes the details of the **set_map()** function under different command word:

Start Map definition

void set_map (*id*,*pbase*)

or

void set_map (*id*,*xbase*,*ybase*)

int *id* ID number of the map to define, ranged from 0 to 15.

POINT *pbase* The base point of the map definition.

double *xbase* X-coordinate of the base point.

double *ybase* Y-coordinate of the base point.

Once a map definition is started, you may use **plotent()** function to generate the drawing map. If this function is issued while in the midst of a map definition, it will terminate the current map definition before starting the new one.

Note that, since the display list of the map definition is in a temporary construct, defining a map of ID number N, will destroy the map definition with ID number large than n. Also, the use of certain Prime Commands will definitely destroy the current map definition in the display list, such as MOVE/COPY..., which involves dragging of objects.

End Map definition

void set_map (*-1*)

You must terminate the creation of map definition by this function call. Issue of **command()** will also terminate the current map definition.

Set Scale Factor for sub-sequent put_map() operation

void set_map (*-4*,*pscale*)

or

void set_map (*-4*,*xscale*,*yscale*)

POINT *pscale* The X/Y Scale factor given as a point.

double *xscale* The X Scale factor.

double *yscale* The Y Scale factor.

This function call setup the scaling factors for the subsequent **put_map()** operations. The scaling factors, one for the X-axis and one for the Y-axis, will be converted into 16-bit integers with a scaling of 256. That is, the range of the scale factor will be from 1/256 to 255.99 with a resolution of 1/256.

A scaling factor of zero will disable the scaling function. So, to turn off Scale Factor Function, issue **set_map(-4,p(0,0))** or **set_map(-4,0,0)**.

Set Rotation COS/SIN for sub-sequent put_map() operation

void set_map (-5,pvect)

or

void set_map (-5,cv,sv)

POINT pvect Vector pair of (Cos,Sin) used for the rotation transformation.

double cv The COS value.

double sv The SIN value.

This function call setup the rotation factors for the subsequent **put_map()** operations. The values of the Cos/Sin will be scaled up by 14 bits and then converted into 16-bit integers. So, its range must be within (-1,1). Use the scale factor function if the transformation is not a pure rotation.

To turn off Rotation Function, issue **set_map(-5,p(0,0))** or **set_map(-5,0,0)**.

Special Notes on set_map() function calls

Once a scaling factor or a rotation Sin/Cos is setup, it will remain effective until it is reset by another **set_map()** function call. So, it is recommended to reset them in the beginning of a program that uses **set_map()** and **put_map()** functions, and reset them before the program exits.

A bug had been located in **set_map(-4,...)** function calls that will not set the scale factor correctly. A work-around is to call the function by

```
set_map(-4,P(xs,ys),P(xs,ys));
```

and to turn off the scaling by

```
set_map(-4,P(0,0),P(0,0));
```

This bug has been fixed at the time this document is updated.

```
Example:      #define delay(n)  for(i0=n;i0>0;i0--);

main()
{
    e1 = gete("\nSelect Object To demo: ");
    set_map(0,e1.pick); // Start map
    plotent(e1); // Draw map
    set_map(-1); // End map
    userbrk(0); // Disable user break
    setplot(0x89,0,0,0); // Set color XOR/Blue
```

```

v3 = 10;    // Rotation Step
v4 = 0.05; // Scaling step
p1 = sin(0),cos(0);
v1 = 0;
v2 = 0.05;
for(;;)
    for(i1=0;i1<=36;i1++)
        set_map(-5,p1); // Set rotation
        set_map(-4,v2,v2); // Scaling
        put_map(0,e1.pick); // Draw it
        p1 = sin(v1),cos(v1);
        v2 += v4;
        v1 += v3;
        delay(10);
        put_map(0,e1.pick);
        if ( userbrk(-1) )
            set_map(-5,0,0); // Turn it off
            set_map(-4,0,0);
            exit();
        }
    }
v3 = -v3;
v4 = -v4;
}
}

```

set_mask

Setup selection mask.

void set_mask (type,ptnname,state)

int type Type of mask to setup. Always 0 at current version for layer mask.

char ptnname[] String, match pattern of layer name. A null string ("") means all layers. See also **strmatch()** for the use of wildcat character in the name match pattern.

int state Mask status, 1 to enable the masking, 0 to disable.

Use this function to setup the layer mask, enable the masking by accessing the **@SNAPFLAG** directly (by OR-ing with 0x2000), and then use **getesel()** to make the selection. See also **getesel()**.

set_mvtrns

Set up axonometric transformation matrix for subsequent **mv_trns()** function calls.

void set_mvtrns (vx,vy,vz,ex,ey,ez)

double vx,vy,vz The X/Y/Z components of the viewing direction vector, respectively.

double ex,ey,ez The X/Y/Z components of the entity plane normal vector, respectively. These are optional arguments. The defaults are [0,0,1], which is the X-Y plane.

The specification of the viewing direction vector and the entity plane normal vector follows the same rules for the MVCOPY command.

Once the transformation matrix is setup, you may use **mv_trns()** to do the axonometric transformation.

set_trns

Set up 2-D transformation matrix for subsequent transformation.

int set_trns (pbase,pofst,pscale,range)

POINT pbase Base point for scaling/rotation.

POINT pofst Translation Offset.

POINT pscale X/Y Scale factor.

double range Rotation angle in units of degree.

or

int set_trns (mirlin)

LINE mirlin Mirror line, setup Mirror operation by a mirror line.

Return Transformation Flag. (currently useless to TCL programmer).

Once the transformation matrix is setup, you may use **ent_trns()** to do the geometry transformation. There are also some other functions that will be affected by the **set_trns()**, like **gextent()**, **ent_copy()**, etc.

The transformation of a data point with regard to the above argument will be processed as in the following order:

1. Taking *pbase* as base point for scaling first.
2. Taking *pbase* as base point for rotation.
3. Translate the point by *pofst*

Note that a scale factor of zero value in one axis, will disable the scaling operation over the axis. Likewise, a zero rotation angle or offset value will disable the respective function as well.

To turn off **set_trns()**, use **set_trns(0)** directly.

set_undo

Specify the undo information.

int set_undo (option)

int option Integer value specifying the undo options. Currently supported options are:

- 0** Specifying to directly regenerate the drawing when UNDOing the result from current running TCL program. If this is not specified, the undoing operation will be done step by step, depending on how the drawing database was modified. If the TCL program may heavily modify the drawing database, this setting will be essential, since the undoing operation may result in slow performance in the graphic display as the **TCAM Graphic Runtime** are modifying the display lists while the display lists are already very large. Regeneration of the drawing in such a case to refresh the display lists will be faster.
If this should be specified, it is better to specify it at the beginning of the program. Note that if your program does modify the drawing data base very much, do specify this option.
- 1** Specifying to mark an undo section. The undo data generated by a TCL application can be subdivided into several undo sections. The TCL application may issue the **UNDO** command to undo each of these undo sections one by one. This option is then used to mark the end of an undo section and also start of another undo section.
Note that the issue of **command()** will force to mark an undo section, and also the execution of a undo-able prime command always forms an undo section, provided that it is successfully executed.
Note also that the **REDO** command is not supported within a TCL application.
For version after V3.0/R3, this function call will return an ID number of the section mark, which can be passed to the new **undo()** function to undo the drawing database to the marked point.
- else** No meaning.

or

int set_undo (varname)

char varname[] String constant of a system variable name, specifying to save the named variable's content to the Undo buffer, such that it may be restored by the undo operation. This should be called for each system variable once and only once before modifying its content, provided that the modification will not be restored later by the program itself, and the restoration of the variable by the Undo operation is essential.

Return -1, if the system variable is not valid, else if the operation is succeeded.

The TCL has incorporated the undo capability behind the programming language. However, since a TCL program may read/write access the system variables as freely as it does to the TCL local variables, it is therefore not adequate to store the undo information automatically each time a system variable is write-accessed. It then turns to be the responsibility of the TCL program to make decision on what to be saved.

Usage Note on set_undo(1):

When a TCL application is started, an Undo Group is also started to account for the recording of the change that the TCL application may do to the drawing database. When the operator issue Undo command from the command line, the whole group is undone, no matter how many changes has been made in there.

The Undo Group is further subdivided into undo sections, depending on how the TCL application is doing. The undo command may also be invoked by the TCL application via the **command()** function call to undo each of the section, of course, successively.

An undo section is formed naturally by the **command()** function call that calls the system kernel to execute a Prime Command. However, as the TCL also provides direct function calls to create entities and modify drawing database, the effects done by these function calls must also be managed by the Undo mechanism. The **set_undo(1)** function call is thus used to specify how these functions calls will form an undo section.

The following example program shows how it works:

```
main()
{
    ... // Setup L1-L8
    newent(L1); // No section formed yet,
    newent(L2); // these are guarded.
    set_undo(1);
    newent(L3);
    newent(L4);
    command("LINE p1 p2 ");
    newent(L5);
    newent(L6);
    set_undo(1);
    newent(L7);
    newent(L8);
    command("u"); // Undo L7 & L8
    command("u"); // Undo L5 & L6
    command("u"); // Undo LINE p1 p2
    command("u"); // Undo L3 & L4
    command("u"); // No effect! -- no more undo sec.
    command("u"); // No effect!!
}
```

Note that in the above, the L1 and L2 can not be undone by the last **command("u")**, since they were not enclosed in any undo section. They belong directly to the Group itself. The TCL application can not possibly undo the whole group by itself. There are Group information that must be guarded. To make L1 and L2 unguarded, in the above example, simply add **set_undo(1)** before them.

A new function **undo()** is added after V3.0/R3 to support the undo operation directly. See **undo()** function for details.

setdata

Initialize data memory with data taken from specific source memory.

long setdata (dst,src,index)

Var <i>*dst</i>	Pointer expression of the destination memory to which the source data is copied.
Var <i>*src or src</i>	Pointer expression of the source memory or direct argument value from which the initialization data is taken.
int <i>index[]</i>	Optional integer array to specify how the source data is copied to the destination memory. See description below.
Return	Number of data byte copied to the destination memory.

This function is used to initialize a data memory by copying existing data from a source memory to that destination memory. It works in a very different way from the **memmove()** function, but is much more versatile. In fact, it can be used directly to replace the both **memmove()** and **memset()** functions.

Two distinct ways to copy the source memory to the destination memory are supported and are specified by the presence of the third argument, *index[]*. If this argument is not present, the source memory is simply copied to the destination memory, byte by byte, without any data conversion and re-ordering. If the size of the destination memory is larger than the source data, the source data will be re-cycled to fill over the destination memory. The only exception is that, when both the source and destination data are of STRING type (character array) and the destination is an array of STRING, **setdata()** will align the source memory to the destination memory in terms of STRING.

If the third argument is present, it must be an integer or an integer array of any size that contains a sequence of index number. An integer value is taken as a single element array containing that value. This sequence of number will be cycled through and used to access the source memory as a subscript index (modulus-adjusted) to the source array in the given data type. The data so accessed will be converted and copied to the destination memory, element by element, according to its data type, until all of it is filled over.

setplot

Setup plotting attributes for **plotent()** function to work.

void setplot (*color, style, state, height*)

int <i>color</i>	Color number of the next entity to plot. Bit 7 is on if plots by XOR-ing.
int <i>style</i>	Style number of line generation. Style 0 is solid line, style 1 is dotted line, while style 2 to 15 are different kind of dash lines.
int <i>state</i>	Integer flag value specifying the plot options regarding the display list control: <ul style="list-style-type: none">Bit 0: Turn display list storage mode ON/OFF. To register the plotting to the display list, set this bit flag value to 1.Bit 5: High-light entity mode ON/OFF, effective only to entity handle or SLIST passed to plotent(). If this bit flag is ON, the drawing entities will be plotted with high-light attribute.Bit 6: If this bit flag is set, the entities will plotted with their own property control and will be registered to the display list. It is effective only to entity handle or SLIST passed to plotent().

Bit 7: If this bit flag is set, the entities will be cleared from the graphic screen and removed from the display list. This bit flag override the bit 5 and bit 6, and is effective only to entity handle or SLIST passed to **plotent()**.

double height Set the elevation of the objects for subsequent 2-D objects plotting.

show_help

Display the help content of a specific TCAM HELP file starting from a specific label.

int **show_help** (*hlpfile*, *label*)

char hlpfile[] Name of the help file to display. If no file extension is given (without the dot), then ".HLP" will be assumed.

char label[] Optional string specifying the label in the help file where to start displaying. If it is not given or not found in the file, the display will start from the very beginning of the file; otherwise, it starts from the specific label.

Return -1, if the specified help file is not found.

This function will search through the paths specified in the environment variable TCADPATH for the specified file, if it is not found in the given path.

This function is useful for a TCL application to provide its own help package in an easy way. See Appendix of Command Reference for details of the TCAM HELP file specification.

show_icon

Setup array of icons in display

int **show_icon** (*wnd*, *iconspec*, *rowno*, *colno*, *dist*)

int wnd[] Integer array of window construct. See **wndopen()** for details description.

char iconspec[][] Array of character strings containing the icon specifications. The number of icon is determined by the number of strings passed by the array; however, a null string in the array will terminate the counting of the icon number.

int rowno Optional row counts of tilts. See description.

int colno Optional column counts of tilts. See description.

int dist Optional distance factor between tilts.

Return Number of screen item created.

This function will setup an array of icons (as screen items) and return the number of items it creates.

A mother window should have been opened before this function call. The *wnd[]* should be totally contained by this mother window, since the area covered by the *wnd[]* can not be restored. All rules about the layout of the icons are the same as **sel_icon()**. See **sel_icon()** for further details.

An equivalent implementation of **sel_icon()** function without the title string is given below:

```
wndopen(iconwnd);  
show_icon(iconwnd, iconspec, rowno, colno, dist);  
i = wnd_gbtn();  
wndclose();  
return i;
```

where the *iconwnd[]* is given by system initial file setup.

sin

Intrinsic function to calculate the trigonometric sine function of the argument value, where the argument is taken to be in units of degree.

double sin (*angle*)

double angle Angle value in units of degree.

Return Sine value of the given angle in double.

Note that the argument is **in units of degree!** It is the **sinr()** function that takes the angle in radians. This is different from the one in Standard C library. The reason for such implementation lies in the fact that the **sin()** function may be called in the expression from the command line input, where the operators are using the degree for angle specification.

sinr

Intrinsic function to calculate the trigonometric sine function of the argument value, where the argument is taken to be in radians.

double sinr (*angle*)

double angle Angle value in units of radian.

Return Sine value of the given angle in double.

or

POINT sinr (*z*)

POINT z Complex number.

Return Sine value of the given argument in complex number.

sort

Sort array of data in ascending or descending order.

int sort (*base, nemt, flag, index*)

Var base[]	Array of objects to be sorted. Valid objects are integer, long, double, POINT, CIRCLE, STRING and ENTITY.
int nemt	Number of element to sort.
int flag	Optional control flag, specifying how to sort, as described below: Bit 0: 1, sorting in descending order, 0, in ascending order. Bit 1: 1, sorting by Y coordinate, 0, by X coordinate. Bit 2: 1, sorting by radius, 0, by center coordinate, effective to CIRCLE data only. Bit 7: If this bit flag is set, and the passed data type is ENTITY, CIRCLE or POINT, the sorting will be done such that a relative shortest path of first level starting from the first entity is returned (traveling each pick point of the entity handle). Else Reserved, should be zero. The default of flag is 0.
int index[]	Optional integer array to return the change of the data ordering after sorting. If this argument is given, it must be an integer array with a size of at least <i>nemt</i> elements. Corresponding to each element of the data array being sorted, each element of this integer array will contain an index number indicating the original ordering of it. This is a new feature supported only after V3.0/R3d. To test if this feature is supported, simple sort an array and give an explicit element number as 1 or 0 (so that there will be no sorting at all). If the return value is not zero, the feature is supported.
Return	0, if the sorting is successful, 1, if the sorting fails, and 2, if the data type is not supported.

The sorting may fail due to any one of the following reasons:

- **1.** The array is too large to be held as a whole in a temporary memory buffer for the quick sort to work.
- **2.** The worse case of the quick sort occurs and the working stack overflows. The array may be partially sorted. This is a rare case.
- **3.** The data type of the array is not supported. Currently, only integer, long, double, POINT, CIRCLE, ENTITY and STRING array can be sorted.

If the total size of the array is less than 1024 bytes, this function will use the stack memory as the buffer to sort it directly. Otherwise, the system will try to release some memory for use as the temporary buffer. The maximum size shall not exceed 64 KB in current release.

If the passed data type is ENTITY, it is sorted by its pick point coordinates.

sqrt

Intrinsic function to calculate the square root of the argument.

double sqrt (*value*)

double value Value to find the square root.

Return Square root of the argument.

or

POINT **sqrt** (*z*)

POINT z Complex number to find the square root.

Return Square root of the argument in complex number.

srand

Initialize the random number generator with new seed.

int **srand** (*seed*)

int seed Seed value, Optional.

Return Actual seed value used for the random number generator.

If no argument is given, this function will automatically initialize the random number generator with a seed taken from the timer.

The random number generator will be initialized by the system timer at least once at the time **TwinCAD** is initialized.

srch_tag

Create specific tag entity and append it to a given entity.

ENTITY **srch_tag** (*ent,tag*)

ENTITY ent Entity handle of a given entity, of which the appended tag-list is searched for the specific tag. If this given entity is a tag entity, it must be a part of a tag-list, and the search will continue from the next tag appended after it.

char tag[] Character string of the tag name, of which only the first 11 characters are taken, and which will be converted to upper case.

Return Entity handle of the first matched tag entity. Null handle if the specific tag is not found from the tag-list appended to the given entity, or there is no tag-list appended.

It is possible and allowed that a tag-list may contain multiple tags of same tag name. To locate each of them, successive calls to **srch_tag()** with the returned entity handle will continue to travel along the tag-list.

sscanf

Parse and read formatted data from string.

int **sscanf** (*s,format[,argument]...*)

char s[]	Source string of formatted text to parse.
char format[]	Format-control string.
Var *argument	Optional arguments of pointer to variables of types specified in the format-control string.
Return	The number of fields that were successfully converted and assigned.

This function performs parsing of formatted text, reading characters from the string given by the first argument, while using the second string argument as the format control string. Additional arguments may then be required, depending on the contents of the control string. Each argument after the control string must be a pointer, pointing to variables where the converted data are stored into. (Though TCL V1.0 does not yet support explicit declaration of a pointer type, the use of pointer expression in the form **&var**, however, is supported.)

The operation of this function can be taken as the reverse processing of the **sprintf()** or **printf()**. Starting from the beginning of the source string to read, the corresponding contents of the format control string may have the following three type of parsing actions:

✧ **Whitespace characters**

A whitespace character in the format control string tells the parser to read and discard the corresponding whitespaces from the source. The reading stops at the first non-whitespace character encountered, which will still remain as the next character to be read from the input source. In short, It is used to skip the possible whitespace characters from the source. Note that if several consecutive whitespace characters appear in the control string, the effect is the same as if only one had appeared.

✧ **Conversion specifications**

A conversion specification tells the parser how the corresponding source string to be read. It begins with a percent sign '%', followed by:

- ⇒ **An optional asterisk '*'**, to specify the reading result will be discarded. If this is present for a conversion operation that normally performs an assignment of the result to the argument, the assignment will be suppressed, and no pointer argument will be consumed. However, the source string will still be read as the conversion specification specifies.
- ⇒ **An optional maximum field width**, expressed as an unsigned decimal integer. This width value must be positive and non-zero. If this field is present, the source string will be read as the conversion operation required to the specific length of characters. If this field is not present, the conversion operation may take the reading from the source as many as possible.
- ⇒ **An optional long size specification**, expressed as the character 'l' (lowercase letter L), which is used in conjunction with the conversion operations 'd', 'o', 'u', and 'x' to indicate that the argument is long.

- ⇒ **A required conversion operation**, expressed as a single character, such as 'c', 'd', 'e', 'f', 'g', 'o', 's', 'u', 'x', etc.

A conversion operation produces result of reading from the source string, and if the assignment is not suppressed, the result will be assigned to the next pointer argument and thus one pointer argument is consumed.

✧ **Other characters**

Any character other than a whitespace character or a percent sign must match the next character from the source string. If it does not match, a reading conflict has occurred and the *sscanf()* operation will be terminated.

There should be exactly the right number of pointer arguments, each of exactly the right type, to satisfy the conversion specifications in the control string. Though the TCL function may check for the type and do the type conversion automatically, for portability reason, care must still be taken.

The conversion operation supported are described below:

- c** One or more characters are read and assigned, as many as specified by the field width. No terminating null character is appended.
- s** A whitespace-delimited string is read and assigned. An extra terminating null character is appended.
- b or B** Signed binary conversion is performed.
- d or D** Signed decimal conversion is performed.
- o or O** Signed octal conversion is performed.
- x or X** Signed hex-decimal conversion is performed. The conversion will read the hex string in the form as either '0xhhh' or 'hhh'.
- u or U** Unsigned decimal conversion is performed.
- e, f, g** Signed decimal floating-point conversion is performed.
- S** Skip white spaces and read the next quoted string. The string must start with either a single quote (') or double quote (") character. If the next character, after skipping the white spaces, is neither a single quote nor a double quote, the string will be read as a null string. Otherwise, the character string after the quote will be taken as a result of the reading until the next same quote character or the end of the source string is met. Note that two successive occurrence of the quote characters within the string will be read in as a single character in the string.

If a width value is specified, the string read will be truncated as required.

After version 3.02, if a Long modifier is added before the 's' or 'S' conversion specifier, it will modify the conversion operation as described below:

- IS** The string will be quoted by the next non-white-space character. It does not check whether it is a single quote or a double quote.

Is The string read from the source may also be delimited by comma, colon and semi-colon character, for the %s reading control. That is the comma, colon and semi-colon character will not be included in the string. This is essential for **sscanf()** to read a comma delimited string.

strcat

Append one string to another string. (Macro)

STRING strcat (*s1*,*s2*)

char s1[] Destination string.

char s2[] Source string.

Return The resulting concatenated string.

This function is implemented for portability to standard C library by using macro definition in TCLSTR.H, as below:

```
#define strcat(s1,s2) (s1+=s2)
```

strchr

Find the first occurrence of a character or sub-string from a string.

int strchr (*src*,*sub*)

or

int strchr (*src*,*cc*)

char src[] Source string.

char sub[] Sub-string to find from the source string.

char cc Character code to find from the source string.

Return 0, if the character or sub-string is not found.
else, position index of the character or sub-string from the source string counted from 1.

This function is supported only after V3.1.055.

strcmp

Lexicographically Compare two character strings.

int strcmp (*s1*,*s2*)

char s1[] First string to compare.

char s2[] Second string to compare.

Return -1, if the first string is less than the second string.
 0, if the first string is equal to the second string.
 1, if the first string is greater than the second string.

This function is implemented for portability to standard C library by using macro definition in TCLSTR.H, as below:

```
#define strcmp(s1,s2) ((s1>s2)?1:(s1<s2)?-1:0)
```

strcpy

Copy string from one to another. (Macro)

STRING strcpy (*s1*,*s2*)

char *s1*[] Destination string.

char *s2*[] Source string.

Return The resulting destination string.

This function is implemented for portability to standard C library by using macro definition in TCLSTR.H, as below:

```
#define strcpy(s1,s2) (s1=s2)
```

strlen

Find the length of a given string.

int strlen (*s1*)

char *s1*[] String to find length, null terminated.

Return The length of the string.

This function is implemented for portability to standard C library by using macro definition in TCLSTR.H, as below:

```
#define strlen(s1) len(s1)
```

strmatch

String wildcat pattern character matching.

int strmatch (*str*,*patn*)

char *str*[] Source string to be checked for a match.

char *patn*[] String pattern for matching, see description.

Return 1 if the given string matches with the pattern, 0 if not.

The following wildcat characters can be used to build the string pattern:

- * (Asterisk)** Matches any string, including the null string. It can be used anywhere in the match pattern. For example, `A*` will match anything starting with A, while `A*C`, anything starting with A and ending with C.
- ? (Question)** Matches any single character.
- # (Pound)** Matches any numeric digit.
- @ (At)** Matches any alpha character.
- % (Percent)** Matches any nonalphanumeric character.
- ~ (Tilde)** Matches anything but the pattern. For example, `~A*` will match anything not starting with A, while `~A*C`, anything not starting with A and not ending with C, and `~A*~C`, anything not starting with A but ending with C (as two tildes before C).
- [...] (Set)** Matches any one of the characters enclosed in the character set. For example, `[ACDF10]` will match a character among the A, C, D, F, 1, and 0.
- [~...] (Set)** Matches any character not enclosed in the set.
- (Hyphen)** Used inside brackets to specify a range for a single character. For example, `[A-D]` will match a single character from A to D.
- ` (Reversed Quote)** Escapes special characters (reads next character literally).

strncat

Append up to n characters from one string to another string. (Macro)

STRING strncat (s1,s2,n)

- char s1[]** Destination string.
- char s2[]** Source string.
- int n** Maximum number of characters to append.
- Return** The resulting concatenated string.

This function is implemented for portability to standard C library by using macro definition in TCLSTR.H, as below:

```
#define strncat(s1,s2,n) (s1+=left$(s2,n))
```

strncmp

Lexicographically Compare two character strings up to n characters.

int strncmp (s1,s2,n)

- char s1[]** First string to compare.
- char s2[]** Second string to compare.
- int n** Maximum number of characters to compare.
- Return** -1, if the first string is less than the second string.

- 0, if the first string is equal to the second string.
- 1, if the first string is greater than the second string.

This function is implemented for portability to standard C library by using macro definition in TCLSTR.H, as below:

```
#define strncmp(s1,s2,n)  strcmp(left$(s1,n),left$(s2,n))
```

strncpy

Copy up to n characters from one string to another. (Macro)

STRING strncpy (s1,s2)

- char s1[]** Destination string.
- char s2[]** Source string.
- int n** Maximum number of characters to copy.
- Return** The resulting destination string.

This function is implemented for portability to standard C library by using macro definition in TCLSTR.H, as below:

```
#define strncpy(s1,s2,n)  (s1=left$(s2,n))
```

Note that this definition of **strncpy()** is different from the Standard C library function, but is much useful. To define the standard one, use the following #define:

```
#define strncpy(s1,s2,n)  (s1=left$(s2,n)+(s1-left$(s1,n)))
```

strchr

Find the last occurrence of a character or sub-string from a string.

int strchr (src,sub)

or

int strchr (src,cc)

- char src[]** Source string.
- char sub[]** Sub-string to find from the source string.
- char cc** Character code to find from the source string.
- Return** 0, if the character or sub-string is not found.
else, position index of the character or sub-string from the source string counted from 1.

This function is supported only after V3.1.055.

strrev

Obtain the reverse of a given string

STRING strrev (*src*)

char *src*[] Source string to be reversed.

Return The result of the reversed string.

Note that the original source string will not be modified. A copy of the result is returned.

strrplx

Replace all specific sub-string from a source string with a given new sub-string.

STRING strrplx (*src,old,new*)

char *src*[] Source string.

char *old*[] Sub-string to be replaced.

char *new*[] New sub-string to replace the old string with.

Return The resulting string of the replacement.

If the new-string is not given, or is a null string, effectively all the occurrence of the old-string in the source string will be removed. Note that the original source string will not be modified. A copy of the result is returned.

strskip

Skip undesired leading characters from a string.

STRING strskip (*src, chrset*)

or

STRING strskip (*src, flag*)

char *src*[] Source string from which the leading characters are to skip.

char *chrset*[] Optional character string containing set of characters to skip, default to " \t", which skip the leading blank spaces and TAB characters. A leading '~' (Tilt) code means NOT to the use of the character set. See example for more details.

int *flag* Integer bit flag to specify which type of codes to skip, as described below:

Bit 0: 1, Skip upper case letters [A-Z].

Bit 1: 1, Skip lower case letters [a-z].

Bit 2: 1, Skip digits [0-9].

- Bit 4:** 1, Skip blank space, TAB, CR, LF, VTAB and Form feed characters.
- Bit 5:** 1, Skip punctuation characters.
- Bit 6:** 1, Skip control characters.
- Bit 7:** 1, Skip blank spaces.
- Bit 8:** 1, Skip hexadecimal characters [0-9, a-f, A-F].
- Bit 9:** 1, Skip integer characters [0-9, +|-].
- Bit 10:** 1, Skip floating point characters [0-9, +|-|.].
- Bit 12:** 1, Skip extended characters [0x80-0xff].
- Bit 15:** 1, Take the negative action of the above.

A list of suggested macro definitions are given below:

```
#define SKP_UPPER      0x1
#define SKP_LOWER     0x2
#define SKP_DIGIT     0x4
#define SKP_SPACE     0x8
#define SKP_PUNCT     0x10
#define SKP_CONTROL   0x20
#define SKP_BLANK     0x40
#define SKP_HEX       0x80
#define SKP_INT       0x0100
#define SKP_FLOAT     0x0200
#define SKP_EXT       0x0800
#define SKP_NOT       0x8000
```

Return The resulting string after skipping the leading characters from the given source string.

A few examples are given below:

```
str=strskip(str," \t"); // skip leading spaces and tabs
str=strskip(str,"~ ,"); // skip leading char until space or
comma
str=strskip(str,0x0100); // skip leading integer number
str=strskip(str,3); // skip leading alphabets
```

This function is supported only after V3.1.055.

strword

Parse a leading word from a given string

STRING strskip (*src*, *chdlmt*)

or

STRING strskip (*src*, *flag*)

char <i>src</i>[]	Source string from which the leading word is to parse.
char <i>chdlmt</i>[]	Optional character string containing set of delimiting characters, default to "\t", which parse the leading word until a blank space and TAB character. A leading '~' (Tilt) code means NOT to the use of the character set. See example for more details.
int <i>flag</i>	Integer bit flag to specify which type of codes to parse, which has the same definition as that in strskip() . See also strskip() .
Return	The resulting string reading from the leading characters of the given source string.

A few examples are given below:

```
str=strword(src," \t"); // Read leading word until space and tab
```

This function is supported only after V3.1.055.

style

Check text style, or load new text style, or read style data.

STYLE style (*name,type,shapefile*)

char <i>name</i>[]	Name of text style to query or to create.
int <i>type</i>	Type of text style. 0 for small font, 1 for extended font.
char <i>shapefile</i>[]	Optional shape font file. If this argument is omitted, than, the first two argument is used to test and to retrieve the STYLE table entry. If this argument is present, and the given name of style is not yet defined, the shape file will be loaded to define the named style.
Return	A table entry of STYLE type. If the style entry is not found, the first entry at the STYLE table will be returned.

This function can be used to load a text style from a disk font file, either from a TCL program or directly from the command line (or from a menu script) without popping up the text style window. However, if the specific style had been loaded in the system, this function will not attempt to reload the text style and so as to redefine it. It simply returns the table entry of that style.

Note that the specified name of style will become current default text style, provided that it has been well loaded.

Another use of this **style()** function is given below:

STYLE style (*id*)

int <i>id</i>	ID number of the specific style, as that returned by styleid() . The range of this ID number is from 0 to 255.
Return	A table entry of STYLE type. If the style entry of the specified ID is not found, the name field of the return STYLE data will be null.

styleid

Obtain specific text style's internal ID number.

int **styleid** (*name*)

char *name*[] Name of the text style to query.

Return Integer value representing the internal ID number of the specified text style. Valid ID number of a loaded text style is from 0 to 255, while a value of -1 in return indicates that the text style is not known to the system. All else negative values represent ID numbers for those text styles defined but not properly loaded yet.

To directly modify an existing TEXT entity's text style reference, you must obtain the text style's ID number first by this function call. Except the value of -1, which can be used directly to indicate no text style being used, no other values than those returned from this function call can be used for the purpose.

sub

Intrinsic function to implement the '-' (minus) operation.

Var **sub** (*args...*)

Var *args* Variable number of arguments of applicable data type. The minimum number of arguments are 2, while the maximum number are 15. Type conversions are performed automatically from left to right in sequence.

Return Result of the subtraction, depending on the type of arguments.

This is the intrinsic function to implement the '-' operation in the expression. In other words, the expression: a-b-c-... can be written as SUB(a,b,c,...). Note that both upper case and low case of the function names are accepted. The type of operands can be integers, doubles, POINTs, and character strings. Type conversions from integer value to double are performed automatically when they are mixed in the operation. See the table below:

Operand1	-	integer	double	string	POINT
Operand2		-----			
integer		integer	double	error	error
double		double	double	error	error
string		error	error	string	error
POINT		error	error	error	POINT

Note that the result of string subtraction will be the removal of the first occurrence of the second string operand.

Also note that the actual implementation of **SUB(a,b,c,...)** is **SUB(a,ADD(b,c,...))**.

swap

Swap the content of two variables.

Var swap (ptr1,ptr2)

Var *ptr1 Pointer expression to a single variable of which the content will be swapped with that of the other variable.

Var *ptr2 Pointer expression to another single variable of which the content will be swapped with that pointed by the first argument.

Return The value pointed by the second argument.

The data type of the two arguments must be the same. The size of data to swap is determined by the data type. Note that this function can not swap the whole of an array. You must address each element to swap one by one.

symbol

Load a specific symbol from specific symbol library.

int symbol (sname,libfile, defsize)

char sname Symbol Name, will be converted to upper case.

char libfile Symbol library file, Optional.

LINE *defsize Optional data pointer to return the base definition size of the symbol (see descriptions).

Return 0, if symbol not found, loading fails; otherwise, the symbol is already loaded and the total number of loaded symbol in system is returned.

This function is used to load a specific symbol from external symbol library. If the specific symbol is already loaded when this function is called, nothing will happen but the current number of loaded symbol is returned; otherwise, the system will try to load the specific symbol from the specified library file. Details operations are given below:

If the specific library file is given:

- 1. The library file is searched first from current directory, through the path specified in **TCADPATH**. If the library file can not be found, the loading fails.
- 2. Search the symbol from the given library. If the symbol can not be located from the library, the loading fails.
- 3. Load the symbol and return total number of symbol loaded in memory.

If no specific library file is given:

- 1. Search through the symbol libraries that have been referenced for symbol loading, for the specific symbol. If the symbol is found, load it and return ok.
- 2. Search through the directory path from current directory through the path given by **TCADPATH** for all the symbol libraries (*.SMB) for loading the specific symbol. If the symbol is found, load it and return ok; otherwise, fail the loading request.

If the third argument is given, it must be a pointer to a LINE data. It is used to return the base definition size of the symbol if the symbol is already loaded in the system. The base definition size of a symbol is defined by the minimum extent point and the

maximum extent point of the symbol's bounding box relative to its insertion base point. Note that the bounding box of a symbol is not necessarily exactly the same as the geometry extent of the symbol itself.

symlib

Create a new symbol library or update the description string of an existing symbol library.

STRING **symlib** (*libfile*, *description*)

char libfile Name of the symbol library file to create or to update.

char description Optional description text for the symbol library.

Return The description text of the specified symbol library file if it is found and valid. Null string is returned if the file is not valid or missing.

If the description text string (the second argument) is given and is not a null string, the symbol library file will be updated with the new description text. If the library file does not exist, **symlib()** will create it with the given text.

However, if the description text is not specified or it is a null string, **symlib()** will not update nor create the symbol library file, but return the description text read from it, provided that it is present.

Note: **symlib()** will not search through the TCADPATH for the specific symbol library file if it does not exist in the specified pathname.

sysdrive

Obtain the informations of the disk drives currently attached to the system.

int **sysdrive** (*diskid*, *rescan*)

int diskid[] Integer array to hold the disk drive informations.

int rescan Optional, specifying whether to re-scan the system for new drive informations. If it is 1, the function will actually re-scan the system for the drive information; otherwise, a pre-scanned drive table information is returned.

Return Number of disk drive in system reported by DOS.

If a value of N is returned, then each array element of the *diskid[]* with a subscript value from 0 to N-1 will contain the information of a corresponding disk drive, as described below:

Bit 0-4 Drive ID number from 0 to 31, in corresponding to drive A: B: C: ... etc., respectively

Bit 5-9 Reserved, should be zero.

Bit 10 1 if drive is CD-ROM

Bit 11 1 if drive is a RamDisk

Bit 12 1 if drive is one created by SUBST

- Bit 13** 1 if drive is one created by JOINT
- Bit 14** 1 if drive is removable
- Bit 15** 1 if drive is a network drive

At start-up, the system will scan all the drive informations and store them in an internal table, and use this table information whenever the drive informations are required. For example, the file window will use these pre-scanned drive informations for the drive icons, instead of scanning the machine each time it is entered. Therefore, in case that a new drive is added or removed during the editing session (e.g. SUBST drives), the system may not reflect it immediately.

To force the system to re-scan the drive and rebuild this table, the TCL application must issue this **sysdrive()** function with the additional parameter set to 1.

SystemMode Win

Return the current system operation mode/level value.

int SystemMode (*regno*)

int *regno* Optional, authentication register number to read. If this argument is given, it returns the authentication value read from that register. The meaning of the returned value depends. This function call is reserved for system use only.

Return If no argument is given, it returns an integer value specifying the current system's operation mode and level:

- >0** Standard Mode.
- =0** TestDrive Mode.
- <0** Lite mode, level in absolute value.

sysversion

Return the details system version code string.

STRING sysversion()

Return The system version code string.

The system version code string will be in the format:

Major-Version-Code (Version Notes) Rev\$nn Created-Date

for the DOS version, such as

V2.10c (Professional) Rev\$51 Mon Jun-12-1995

and will be in the format:

TwinCAD Version-Code Created-Date

for the Windows version, such as

TwinCAD V3.1.046 Fri Nov 21, 1997

takename

Extract the name of file from a full path name specification.

STRING takename (*fname*)

char *fname* Fullname of file from which filename is to extract

Return The name of the file.

takepath

Extract the path of a fullname.

STRING takepath (*fname*)

char *fname* Fullname of file from which the path is to extract

Return The path of the file.

If the given filename does not contain a path specification, the path to current directory is returned.

tan

Intrinsic function to calculate the trigonometric tangent function of the argument value, where the argument is taken to be in units of degree.

double tan (*angle*)

double *angle* Angle value in units of degree.

Return Tangent value of the given angle in double.

Note that the argument is in units of degree! It is the **tanr()** function that takes the angle in radians. This is different from the one in Standard C library. The reason for such implementation lies in the fact that the **tan()** function may be called in the expression from the command line input, where the operators are using the degree for angle specification.

tanr

Intrinsic function to calculate the trigonometric tangent function of the argument value, where the argument is taken to be in radians.

double tanr (*angle*)

double *angle* Angle value in units of radian.

Return Tangent value of the given angle in double.

or

POINT tanr (z)

POINT z Complex number.

Return Tangent value of the given argument in complex number.

tbl_count

Count the number of specific type of tabled objects existing in the system.

int tbl_count (type, ptnname)

int type Type of the named objects to count:

- 0** Layer
- 1** Linetype
- 2** Text Style
- 3** Blocks
- 4** User Defined Variables
- 5** Regions
- 6** Tag Definitions
- 7** Grouped Tags
- 8** PUCS (matching pattern not used).
- Else** Reserved, Ignored.

char ptnname[] Optional, match pattern name of the objects. Default to "*". Note that "" means ALL here.

Return Number of the objects found.

tbl_next

Get the next table element from a given one, such as LAYER, LTYPE, STYLE, and so forth.

Var tbl_next (current)

Var current Current table element such as LAYER, LTYPE and STYLE, must be well initialized.

Return The next table element after the given one is returned with the same data type. If there isn't any next table element to index to, the name field of the returned structure will contain a null string.

or

STRING tbl_next (curname, type)

char curname[] Name of current entry in the table.

int tyte Type of the table to travel. See **tbl_count()** for the supported values.

Return The name of the next table entry. Null string is returned if it is already end of table. If the *curname* is not found in the table, the name of the first entry will be returned.

Example: LTYPE myltype;

```
int i, idx;
myltype = getltype(); // get first entry from table
for(i=0;myltype.name!="";i++){
    printf("\n%s",myltype.name);
    myltype = tbl_next(myltype);// obtain next entry
}
printf("\nTotal %d linetypes loaded.",i);
```

text_len

Calculate the effective display length of a text string.

int text_len(str)

char str[] Character string of which the display length is to be calculated.

Return Integer number of character position of the effective display length of the given text string.

The text string may contain special control characters that will not be displayed in a TEXT, such as '~'. This function is used to count the effective characters number from a given string that will be displayed in TEXT.

text_span

Calculate the actual total display length of a given string under current TEXT style setup.

double text_span(str)

char str[] Character string to be calculated.

Return Total display length of the TEXT string under current text style setup.

tg_close

Close the Application's Text Generation File

void tg_close()

tg_id

Obtain the current Application Text Generation's ID Number

int tg_id()

Return -1, if no TG loaded, else the ID number of the loaded TG.

tg_open

Open and load the Application's Text Generation File

int tg_open(fname,id)

char fname[] Name of the TG file to load in.

int id Optional TG ID number for loading check.

Return -1, if the loading fails, else the ID number of the TG file loaded.

This function will open the specific TG file and load it into system. If the ID number is given, and is not zero, the TG's ID number will be checked. If the id number does not match, the opening fails. If a TG file has been opened previously, this function will close it first.

The file name of current opened TG file is stored in system variable: **TG_FILE**.

About the TG file

The **TG** file is a Text Generation file for a TCL application, so much the same as the **TGF** for **TwinCAD**.

Once a TG file is opened, it will be accessible to all subsequent TCL applications until it is closed explicitly by **tg_close()**. The system will not close a TG file automatically upon the termination of a TCL application.

TG file is important to TCL applications of international interest. Each TCAM optional product built upon **TwinCAD** may have its own TG file to associate with in the future. The **TGF** file will be used only for the Master Products such as **TwinCAD**.

A utility named **TGTRNS.EXE** is supplied to transform a TG file to a common text file for editing purpose, and from a text file to TG file. A document on TGTRNS.EXE will be given separately, in which the structure of a TG file is also discussed.

The **tg_open()** will accept a TG file in both transformed and non-transformed format (ASCII text). However, shipping the TG files in transformed format will prevent the curious end-users from modifying them accidentally.

tg_read

Read text from the Application's Text Generation File

STRING tg_read (txtid)

int txtid Text ID number to read from the TG file.

Return Text string read from TG. Null string returned if no TG file active or the text with the specific ID number is not found.

This function is used to retrieve text information from the TG. All the texts are identified by a text ID number.

tick

Obtain current system timer tick count (INT 0x1A).

long tick()

Return The system timer tick count returned by BIOS interrupt 0x1A.

Note that the timer counts in 18.2Hz, i.e., approximately 18 ticks counts for a second.

time

Obtain current system time information.

STRING time (*itime*)

int itime[4] Integer array of 4 elements to hold the returning information of current system time, which are in sequence the hour (0-23), minute (0-59), second (0-59) and hundredth of second (0-99).

Return The system time in string as "**hh:mm:ss.tt**".

If it is the system time in string of interest, use the intrinsic function **time\$()** instead. See also **time\$()**.

time\$

Intrinsic function to return current system time in string.

STRING time\$()

Return The system time in string as "**hh:mm:ss.tt**".

You may directly reference the **time\$()** function as a data variable as **time\$**. This function can be called from the command line to show the current system time, as if it were a command.

timer

Precision timing service with the system timer.

long timer (*cmd*)

int cmd Optional command word specifying how to read the timer, default 0, as described below:

- 1 Set the start mark on the timer for later reference.
- 0 Return the elapse time since the system start-up in milliseconds.
- 1 Return the elapse time since the start mark of the timer in milliseconds.
- 2 Return the elapse time since the last reading of the timer in milliseconds.

- 3** Reserved, same as 2.
- 4** Return the elapse time since the system start-up in micro-seconds.
- 5** Return the elapse time since the start mark of the timer in micro-seconds.
- 6** Return the elapse time since the last reading of the timer in microseconds.
- 7** Reserved, same as 2.
- Else** The lower 3 bits are used as the command from 0 to 7.

Return Long integer of the elapse time in unit of milliseconds or micro-seconds, depending on the command.

The system will initialize the 8253 timer 0 with mode 2 (divide-by-N) upon system start-up, to enable precision timing using the timer. The change of the timer working mode (the BIOS use mode 3) shall not affect the BIOS timing interrupt anyhow. Nevertheless, the user may disable this mode change by setting the system environment variable "TIMER" to 1.

The precision of the timing is about 0.83 s. However, the overhead of the calling function will not be taken into account for the timing result.

If the precision timing is disabled by the environment variable "TIMER", the precision of the timing may be at best in 27.5 ms, since the standard BIOS will set the 8253 timer 0 to work at mode 3.

If the OS is Windows, TwinCAD will not read the 8253 timer directly, but call Windows API to read the time tick, and the best precision of the timing may be at best to 1 ms.

ToBig5

Convert Uni-coded string to Big5 coded string.

STRING ToBig5(char str[])

char str[] Uni-coded string to be converted to Big5.

Return Converted string.

This function assumes that the input string is in Uni-coded Chinese.

ToGB2312

Convert Uni-coded string to GB2312 coded string.

STRING ToGB2312(char str[])

char str[] Uni-coded string to be converted to GB2312.

Return Converted string.

This function assumes that the input string is in Uni-coded Chinese.

ToKSC

Convert Uni-coded string to KSC coded string.

STRING ToKSC(char str[])

char str[] Uni-coded string to be converted to KSC.

Return Converted string.

This function assumes that the input string is in Uni-coded Korean.

ToolBar

Manipulating the Toolbars.

int ToolBar()

Return The total number of toolbars created in system.

int ToolBar(name, state)

char name[] Name of specific toolbar.

int state Optional integer, specifying new status of the toolbar.

Return The current status of the specific toolbar given by name.

STRING ToolBar(which)

int which Integer, specifying a creation order number of toolbar.

Return The name of the toolbar in the specific creation order. NULL, if the toolbar does not exist.

ToSJIS

Convert Uni-coded string to SJIS coded string.

STRING ToSJIS(char str[])

char str[] Uni-coded string to be converted to SJIS.

Return Converted string.

This function assumes that the input string is in Uni-coded Japanese.

tx_close

Close a text line record R/W buffer.

int tx_close(id, fname)

int *id* Integer ID of the memory buffer handle returned by **tx_open()**.

char *fname*[] Optional name of the text file to save from the buffer. If this argument is given, the text content in the specified record buffer will be saved to the specified file before closing the memory buffer handle.

Return 0, if the memory buffer handle ID is not valid. 1, if the memory buffer handle is released successfully. If the optional text file is given to save the buffer content, it returns the number of text lines written out.

tx_delete

Delete specific range of text record from the record R/W buffer.

int tx_delete(*id*, *start*, *end*)

int *id* Integer ID of the memory buffer handle returned by **tx_open()**.

long *start* Index number of starting record to delete.

long *end* Index number of ending record to delete inclusively. This argument is optional. If it is not given, it defaults to the same index number of the starting record.

Return 0, if the memory buffer handle ID is not valid; otherwise, the total number of bytes removed.

tx_empty

Empty the text record R/W buffer.

int tx_empty(*id*)

int *id* Integer ID of the memory buffer handle returned by **tx_open()**.

Return 0, if the memory buffer handle ID is not valid; 1 if the operation is successful.

tx_insert

Insert text string into record R/W buffer.

int tx_insert(*id*, *index*, *str*)

int *id* Integer ID of the memory buffer handle returned by **tx_open()**.

long *index* Record index number of the string to insert.

char *str*[] String value to insert.

Return 0, if the memory buffer handle ID is not valid; otherwise, the total number of bytes written.

tx_lineno

Obtain the current maximum record number in the R/W buffer.

long **tx_lineno(*id*)**

int *id* Integer ID of the memory buffer handle returned by **tx_open()**.

Return A value of 0 is returned if the memory buffer handle ID is not valid or the buffer is empty; Otherwise, the index number of the last record is returned.

tx_load

Load text file into text line record R/W buffer.

long **tx_load(*id, fname, index*)**

int *id* Integer ID of the memory buffer handle returned by **tx_open()**.

char *fname*[] Name of the text file to load into buffer.

long *index* Optional starting record index number to insert the text file content into the memory buffer. If this argument is not given, the buffer will be cleared first, and the file name will be saved to the record 0, and its content will be loaded starting from the record 1.

The maximum length of a text line record is limited to 255 character excluding the EOL characters (CR/LF). Text line records exceeding this limit will be splitted into successive records at the text file loading.

Return 0, if the memory buffer handle ID is not valid or the file can not be open for reading. Else, the total number of text line read in.

tx_open

Open a text line record R/W buffer.

int **tx_open(*fname*)**

char *fname*[] Optional name of the text file to load in buffer initially. This this argument is not given, the line record buffer will be empty initially.

Return -1, if no more memory buffer handle available. Otherwise, ID of the memory handle of the record buffer is returned.

This function will initialize and allocate a memory buffer for subsequent text line record R/W index access. It provides an easy way to random access a text file of line records with variable record length. The returned ID must be used in the subsequent **tx_xxx()** function calls.

tx_read

Read string from text line record R/W buffer.

STRING tx_read(*id*, *index*)

- int *id*** Integer ID of the memory buffer handle returned by **tx_open()**.
- long *index*** Record index number starting from 0.
- Return** String record read from the buffer. The maximum length of the record is limited to 255 characters.

tx_save

Save text line record R/W buffer to specific text file.

long tx_save(*id*, *fname*, *start*, *end*)

- int *id*** Integer ID of the memory buffer handle returned by **tx_open()**.
- char *fname*[]** Name of the text file to save to.
- long *start*** Optional starting record index number to save to the text file. Defaults to record 1, assuming record 0 contains original text file information.
- long *end*** Optional ending record index number (inclusive) to save to the text file. Defaults to the last record in the buffer.
- Return** 0, if the memory buffer handle ID is not valid or the file can not be created for writing. Else, the total number of text line written out.

tx_search

Search a specific string from the text line records.

int tx_search(*id*, *str*, *start*, *flag*)

- int *id*** Integer ID of the memory buffer handle returned by **tx_open()**.
- char *str*[]** The string to search.
- long *start*** Optional starting record number to search inclusively, default 0.
- int *flag*** Optional integer to specify the searching details, default 0, as described below:
- Bit 0: 0, search forward, 1, search backward.
 - Bit 1: 0, case sensitive, 1, ignoring cases.
 - Bit 2: 0, full record matching, 1, sub-string matching.
 - Else: Reserved, should be zero.
- Return** -1, if the memory buffer handle ID is not valid, else the record number that matches with the search string.

tx_sort

Sort the text line records.

int **tx_sort**(*id*, *flag*)

- int** *id* Integer ID of the memory buffer handle returned by **tx_open()**.
- int** *flag* Optional integer to specify the sorting details, default 0, as described below:
- Bit 0: 0, sort in ascending order, 1, in descending order.
 - Bit 1: 0, case sensitive, 1, ignoring cases.
 - Else: Reserved, should be zero.
- Return** -1, if the memory buffer handle ID is not valid, 1, if the sorting is successful.

tx_write

Write text string into record R/W buffer.

int **tx_write**(*id*, *index*, *str*, *end*)

- int** *id* Integer ID of the memory buffer handle returned by **tx_open()**.
- long** *index* Record index number of the string to write.
- char** *str*[] Optional string value to write. If this text string is not given, it effectively mark the end of the text record buffer; otherwise, the record of the specific index number will be updated by this text string.
- long** *end* Optional record index number (inclusive) to fill with the string starting from the *index* record number.
- Return** 0, if the memory buffer handle ID is not valid; otherwise, the total number of bytes written.

ufloor

Intrinsic function to round the first argument down to a multiple of the second argument (toward negative infinity).

double **ufloor** (*value*, *unit*)

- double** *value* Double value to round.
- double** *unit* Unit value for the rounding.
- Return** Double value that is the largest number not greater than the argument, and that is also an exact integer multiple of the given unit. If the value of the argument is already an integer multiple of the second argument, then the result equals to the first argument.

undo

Direct undo the change made to the drawing database to a specific undo section mark.

int undo (*mark*)

int *mark* Optional, integer ID of undo section returned by **set_undo(1)** function call. If this argument is not given, it will undo to the most recent mark.

Return -1, if nothing to undo (due to incorrect ID mark), 1, if the operation is successful and the specific mark is met, and 0, if the specific mark is not met and everything is undone (should be an impossible case).

With this new function call, the TCL application no longer needs to count for the current undo sections created during a specific application session, so as to undo the operation back to a certain point. An example usage is given below:

```
i = set_undo(1);    // Set a mark here
...
undo(i);           // Undo to the mark!
```

upper

Convert argument string to upper cases.

STRING upper (*str*)

char *str*[] String argument to convert to upper case.

Return A temporary copy of the result (used for assignment).

userbrk

Enable/Disable <Ctrl/C> interrupt from user.

int userbrk (*cmd*)

int *cmd* Negative returns the current <Ctrl/C> state, zero disables <Ctrl/C> interrupt, and positive enables <Ctrl/C> interrupt.

Return 1 if system has been interrupted (<Ctrl/C>), 0 if not.

If the <Ctrl/C> interrupt has been enabled, the user may stop the TCL program execution at anytime by pressing the <Ctrl/C>. If it is disabled, the polling of the user break signal is done only when the application issues the **userbrk(-1)**. The system will not poll the signal automatically until it is enabled again by **userbrk(1)**. This will make significant gain in program execution speed.

The initial default of interrupt state is in enabled state.

uservar

Create/Modify user variables

int uservar (name,cmd,atrb,type,initval)

char name[] Character string of at most 12 characters (including the null terminator), specifying the name of the user variable to operate on. Valid character codes for a user name are alphabets, digits, underline, and dollar sign. All lower case characters will be converted to upper case.

int cmd Optional integer, specifying the operation desired of this function call. If this argument is missing, it defaults to 0, which specifies to check whether the user variable is present or not. Valid operation codes are described below:

VPCHECK

An integer value of 0, specifies to check whether the specific user variable is present or not. The rest of the arguments are ignored in this operation. This is also the default.

VCREATE

An integer value of 1, specifies to create the specific user variable if it is not present in the system. If the specific user variable has already been present in the system, the function will do nothing but return 1, signaling the present of the variable. The creation of a user variable will require the rest of the arguments to specify the attribute, type and initial value assigned to the variable.

VDELETE

An integer value of 2, specifies to remove the specific user variable if it is present in the system. If the specific user variable is not present, it does nothing but return 0. The rest of the arguments are ignored in this operation.

VMODIFY

An integer value of 3, specifies to modify the attribute, optionally the type and current content of the specific user variable if it is present in the system, or to create such user variable if it is not present in the system when this function is called.

int atrb Optional integer, specifying the attribute of the user variable. This argument is required when in the VCREATE or VMODIFY mode. The attribute value is bit-coded. Valid big flags are given below:

VVOLATILE

Bit 0 flag, specifies that the user variable is temporary with the drawing. It will not be saved with the drawing.

VREADONLY

Bit 1 flag, specifies that the user variable is readonly to the operator. It can not be modified by the operator via the **USERVAR** command operation, nor can it be by assignment in expression. The only way to modify the content of the variable with such VREADONLY

attribute is to call this function via the VMODIFY operation mode.

int type Optional integer, specifying the data type assigned to the specific user variable. If this argument is missing, the VCREATE operation will default it to V_INT, and the VMODIFY will ignore it and make no modification on the data type of the existing user variable. Valid data type codes supported are as below:

V_INT

An integer value of 0, specifies that the data type is a signed integer. This is also the default for VCREATE if this argument is missing.

V_ONOFF

An integer value of 1, specifies that the data type is an integer representing logical true/false.

V_COLOR

An integer value of 2, specifies that the data type is an integer representing color number.

V_STRING

An integer value of 3, specifies that the data type is a character string of 32 bytes in length.

V_DOUBLE

An integer value of 4, specifies that the data type is a floating point double value.

V_ANGLE

An integer value of 5, specifies that the data type is a floating point double value representing angle value in units of radians.

V_POINT

An integer value of 6, specifies that the data type is a point coordinate.

Var *initval* Optional argument of variable data type, depending on the data type of the user variable, specifying the initial value or the desired value to assign to the user variable.

Return 1 if the variable is present, and 0, if not.

This function is very useful in that it allows TCL applications to leave global variables in the system so that other applications or successive call to the same TCL application may continue to access and maintain these variables. More than that, these variables may be saved to the disk with the drawing, so that the next time the drawing is loaded, these variables are loaded.

Once a user variable is created, it may be accessed in the same manner as the system variables in the TCL expression by the form "@**varname**". Yet, it is better to check whether a specific user variable is present or not at the first time, before accessing such a variable in a TCL program. Otherwise, the TCL program may receive error message, and the execution will be aborted.

Since a user variable has higher priority over system variable in access, it is possible to create a user variable with the same name as a system variable and thus hide the system variable from the TCL applications. However, a TCL application may easily identify this situation by checking the presence of the specific user variable that has the

same name as the system variable in question. The only way to bypass the user variable and to access the hidden system variable, is to remove the user variable.

v or V

Intrinsic function to return a value by geometry definition.

double V (*argument...*)

Var *argument* Variable number of arguments of predefined type. Different number of arguments of different type will specify different geometry definition of a value.

Return Value obtained by the geometry definition.

The geometry value that can be obtained by this function call, depends on the arguments passed to the function, and are described below:

value	Return the equivalent double value. This is the trivial function.
pnt	Calculate the distance from the origin (0,0) to the given point. The single argument must be of POINT type.
ent	Calculate the length of a given entity. Valid types of argument are LINE, ARC, CIRCLE, and ENTITY. If the argument is an entity handle, the entity it points to will be read in for the length calculation. Valid entities include line, arc, circle, 3d-line, and polyline.
pnt,ent	Calculate the distance from a given point to another given entity. The first argument must be of POINT type. While for the second argument, valid data types are POINT, LINE, ARC, CIRCLE, and ENTITY. The returned distance value will be the mathematically shortest distance between the point and the entity. If the returned value is zero, it means the point is on the entity.

Note that this intrinsic function will be invoked automatically by the pure assignment operator (=) when the data type of the L-value is a scalar value. For example, the expression

```
V1 = P1,P2
```

will be equivalent to the expression

```
V1 = V(P1,P2)
```

where the **V()** function is called by the assignment operator to evaluate the comma expressions to the right of it, where the distance from P1 to P2 is returned.

version

Obtain the version of current running TCL interpreter

int version()

Return An integer value representing the current version of the TCL interpreter. For example, the returned value of 101, means version 1.01, which is the first version of TCL that supports this function.

wide_arc

Calculate the geometry boundary of an arc (as the center line) with a starting and an ending width specification.

int **wide_arc** (*acen*, *pwidth*, *abound*, *type*)

ARC *acen* The circular center line segment.

POINT *pwidth* The width specification in POINT type. The X component stores the starting width, and the Y component, the ending width.

ARC *abound*[] The returned boundary arc segments.

int *type* Optional, generation type. If this value is zero, which is the default, then only on single arc segment is generated for the boundary segment on each side. That is, the right side boundary will be returned in **abound[0]** and the left side, in **abound[1]**.

However, if this value is not zero, then two arc segments will be used for the boundaries on each side. The right side boundaries will be returned in **abound[0]** and **abound[1]**, and the left side, in **abound[2]** and **abound[3]**.

Return An integer value indicating the boundary conditions, as defined below:

- 0** Both the starting and ending width are zero.
- 1** The starting width is not zero.
- 2** The ending width is not zero.
- 3** Both the starting and ending width are not zero.

Given a starting width and an ending width specification for a uniform tube with a circular center line whose center is at the origin, the analytic solution to the boundary curve is a spiral in the form:

$$r = r0 \pm (A*(t-t0)+B)$$

where **r0** is the radius of the center line and **t0** is the starting angle. The coefficient **A** and **B** can be found by the boundary conditions from the given two end points.

However, as the purpose of such shape generation proposed by the system does not require very precise generation of the true spiral, **wide_arc()** provides two simple methods to approximate the shape:

Type 0: Single Arc

A middle point is located from the curve, and an arc is formed by passing the start point through the middle point and to the end point. This is the default. It works nice when the span of the arc is small, say less than 90°. However, if the span angle of the arc is to large, say larger than 180°, the resulting arc may be unacceptable!

Type 1: Spline Arc

Taking the start point and end point as the known boundary conditions, the initial tangent and the final tangent values are also found for a single spline approximated with two arc segments. This works far better than the Type 0, but takes two arc segments instead of one.

For a serious application that requires precise profile of the spiral curve, the programmer may partition the center arc into pieces for the calculation.

wide_line

Calculate the geometry boundary of a line (as the center line) with a starting and an ending width specification.

int **wide_line** (*lcent*, *pwidth*, *lbound*)

LINE *lcent* The center line segment.

POINT *pwidth* The width specification in POINT type. The X component stores the starting width, and the Y component, the ending width.

LINE *lbound*[2] The returned boundary line segments.

Return An integer value indicating the boundary conditions, as defined below:

- 0** Both the starting and ending width are zero.
- 1** The starting width is not zero.
- 2** The ending width is not zero.
- 3** Both the starting and ending width are not zero.

The resulted boundary line segments are returned by the third argument, which must be an array of LINE with at least two elements. The first line segment so returned in the array is always the right side boundary with respect to the given center line segment, and the second one, is to the left side.

winexec Win

Initiate an external program

int **winexec** (*cmdline*, *showflag*)

char *cmdline*[] Text string containing the program name and possible parameters.

int *showflag* Integer control flag to give initial state of the shelled program if it is a Windows Application. This parameter has no effect if the shell program is a DOS Application or is controlled by PIF. Default is 1.

Interested values are listed below:

```
#define SW_HIDE            0
#define SW_NORMAL         1
#define SW_SHOWMINIMIZED  2
#define SW_MAXIMIZE       3
#define SW_SHOWNOACTIVATE 4
#define SW_SHOW           5
#define SW_MINIMIZE       6
#define SW_SHOWMINNOACTIVE 7
```

```
#define SW_SHOWNA      8
#define SW_RESTORE    9
```

Return Negative value, if the function fails and the absolute value of the returned value is the error code. Otherwise, it returns the program ID number maintained by TwinCAD.

This function calls **Windows API WinExec()** directly, to execute an external program. Note that the shelled program by **winexec()** may not terminate or even start to run when this function returns, if the shelling is successful. To check whether the program has terminated, use **is_running()** function. For more informations on **Windows API WinExec()**, see Windows SDK reference.

If it is necessary to wait for the program to terminate, use **exec()** instead.

wnd1btn

Pop up a one-button message box to display message and wait for the user response in GUI mode.

int **wnd1btn** (*subject,message,btn*text)

char subject[] Text string to be displayed in the title bar, usually used to point out the subject.

char message[] Text string to be displayed in the middle of the box, center adjusted, usually containing the message itself.

char btntext[] Optional text string used as the display text on the button in the message box. If this argument is omitted or is of wrong type, a default text string, usually ' OK ', from the system text generation file will be used automatically

Return Codes indicating how the message box is exited:

- 1** The button at the lower center is picked up.
- 0** The [-] button at the upper right is picked up.
- 1** The operator has pressed the <ESC> or <Ctrl/C> key.
- 2** The operator has pressed the <End> key or the middle button of the mouse.
- 3** The operator has pressed the <Home> key or the right most button of the mouse.

This function is used to display a message text by popping up a text window (size, location automatically determined) with a specific subject in the title bar. The GUI mode will be entered for the user to exit the text box display via picking up the appropriate button or pressing specific keys from the keyboard.

Since the main purpose of this function is to display messages in GUI mode, the return value shall be useless in most of the case.

wnd2btn

Pop up a two-button-yes-no query box to display a querying message and then wait for the user response in GUI mode.

int **wnd2btn** (*subject,query,btn1text,btn2text*)

char subject[] Text string to be displayed in the title bar, usually used to point out the subject.

char query[] Text string to be displayed in the middle of the box, center adjusted, usually containing the querying message itself.

char btn1text[] Optional text string used as the display text on the left side button in the query box. If this argument is omitted or is of wrong type, a default text string, usually ' OK ', from the system text generation file will be used automatically

char btn2text[] Optional text string used as the display text on the right side button in the query box. If this argument is omitted or is of wrong type, a default text string, usually ' Cancel ', from the system text generation file will be used automatically

Return Codes indicating the operator's answer:

1 The screen button at the left side is picked up (positive answer).

0 The screen button at the right side is picked up (negative answer).

-1, -2, -3 The operator has pressed the <ESC> or <Ctrl/C> keys.

This function is used to query the user for a Yes-no decision. It pops up a text window (size, location automatically determined) with a specific subject displayed in the title bar, and a query message, center adjusted, in the middle. The GUI mode will be entered for the user to make the decision by picking up the appropriate button or pressing specific keys from the keyboard.

wnd_bitmap Win

Show Bitmap image in current selected window.

int **wnd_bitmap** (*name, atrb, xsize, ysize, tcolor*)

char name[] Name of the external bitmap file or internal ID name. Built-in resource ID number can be given in "#n" format.

int atrb Optional control attribute of the bitmap in display, default 0. Effective bit flags are listed below:

Bit 14: ON, use specified color as the transparent background.

else Reserved, should be 0.

int xsize Optional, required width of the bitmap in pixel, default 0. If this argument is not given or is 0, the original width of the bitmap will be used. If it is given and is not zero, the bitmap will be stretched to the required width.

int ysize Optional, required height of the bitmap in pixel, default 0. If this argument is not given or is 0, the original height of the bitmap will be used. If it is given and is not zero, the bitmap will be stretched to the required height.

long tcolor Optional RGB color value, specifies the transparent color of the bitmap, default 0L (Black). If the argument is given, the bitmap will be drawn using

this color value as the transparent color. If this argument is not given, but bit 14 of *atrb* is set, the black color will be taken as the transparent color.

Return True, if the bitmap is loaded and drawn ok; false, if it fails to load the specified bitmap.

The transparent color of the bitmap means the color value from the bitmap that will be replaced by the background color when it is drawn.

If the name of the bitmap is not given or is not found, **wnd_bitmap()** will create a solid-color fill rectangle and the transparent color will be used to fill the rectangle. This feature is valid only after V3.1.055.

wnd_button Win

Create button control directly.

int **wnd_button** (*row, col, size, atrb, text*)

int row Row specification of the button position. See **wnd_loct()**.

int col Column specification of the button position. See **wnd_loct()**.

int size Size specification of the button. See **wnd_field()**.

int atrb Attribute of the buttons. See **wnd_field()** for details.

char text[] Optional caption text of the button.

Return Button ID (field ID).

This function is equivalent to the following sequence of function calls:

```
wnd_loct(row,col);  
wnd_field(0,atrb,size);  
wnd_puts(str,atrb);
```

However, this function creates the button directly without any overhead in window hit-testing and caption text modification.

wnd_cpos Win

Obtain current cursor position from the current selected window.

long **wnd_cpos()**

Return Current cursor position from current selected window. The lower word of the return value gives the column position of the cursor, while the higher word gives the row position in the column/row specification respectively.

The **wnd_loct()** and **wnd_rloct()** will return the last cursor position in the same format as this function.

wnd_editbox Win

Create a standard Windows Editbox control.

int **wnd_editbox** (*row, col, size, atrb, text*)

- int row** Row specification of the editbox control's position. See **wnd_loct()**.
- int col** Column specification of the editbox control's position. See **wnd_loct()**.
- int size** Size specification of the editbox control. See **wnd_field()**.
- int atrb** Integer, attribute of the editbox control. See description text later.
- char text[]** Optional text to be placed in the edit box initially.
- Return** Field ID.

This function is used to create a Window standard edit box control and is equivalent to the following sequence of function calls:

```
wnd_loct(row,col);  
wnd_field(2,atrb,size);  
wnd_settext(2,str);
```

However, this function creates the edit box directly without any overhead in window hit-testing and caption text modification.

The edit box so created by the **wnd_editbox()** is a Windows standard edit box control. The behavior of such control shall follow the Windows specification about it. The TCL program will not receive any activation from this kind of controls. However, it may use the **wnd_settext()** to retrieve the information maintained by such controls.

Note also the Windows edit box will allocate memory from the local heap which is very little in TwinCAD (16-bit version). So, don't put too many things in the editbox.

The effective bit flag control values for the argument *atrb* word in standard edit box creation are given below:

```
#define ES_LEFT            0x0000   // Left justify  
#define ES_CENTER        0x0001   // Center justify  
#define ES_RIGHT         0x0002   // Right justify  
#define ES_MULTILINE     0x0004   // Allow Multiline editing  
#define ES_UPPERCASE     0x0008   // Upper case only  
#define ES_LOWERCASE     0x0010   // Lower case only  
#define ES_PASSWORD      0x0020   // Password mode  
#define ES_AUTOVSCROLL   0x0040   // Auto scroll vertically  
#define ES_AUTOHSCROLL   0x0080   // Auto scroll horizontally  
#define ES_NOHIDSEL      0x0100   // No hiding of Selection  
state  
#define ES_OEMCONVERT    0x0400   // Convert OEM character  
#define ES_READONLY      0x0800   // Read only, no editing  
allowed
```

See Windows SDK documentation about the details of these control flag meaning.

The following are additional bit flag control values imposed by TCL:

```
#define ES_VSCROLL        0x1000   // Add Vertical scroll bar  
#define ES_HSCROLL       0x2000   // Add Horizontal scroll bar
```

```
#define ES_BORDER      0x4000 // Draw box border
#define ES_LARGE      0x8000 // Enlarge the size by 4
pixels
```

wnd_fgetd

Open a text entry field with a helping screen keypad for the operator to enter or modify a digital value.

double wnd_fgetd (*id, npos, atrb, deflt, row, col*)

- int *id*** Integer number, specifying the screen item field's ID number, where the entry field is to open, provided that it is a valid one (which must be positive, and may start from zero if no title bar is used for the window). If the ID number is not valid for current window, then the field will be opened at the current cursor position.
- int *npos*** Integer number, specifying the width of the entry field.
- int *atrb*** Integer number, specifying the color attribute (text foreground and background color) of the entry field.
- double *deflt*** Optional double value (integer, long will be converted) for the entry field. If this argument is present, the text entry field will be initialized with a default ASCII string which is equivalent to the value.
- int *row*** Optional row position of the upper-left corner to digital keypad.
- int *col*** Optional column position of the upper-left corner to digital keypad.
- Return** Double value that the operator has just entered to the field.

This function is used to open a text entry field, with/without an initial default, for the operator to enter or modify a value. The digital screen keypad is automatically popped up, such that the operator may use the pointing device to enter the value directly.

If a valid screen item field's ID number is supplied, then the entry field will be opened starting from the beginning of the item field. However, the TCL programmer must supply the width of the entry field explicitly, which should be the same as the one of the item field for visual consistency.

The text entry field will be opened at the current cursor position, if the first argument is not a valid item field's ID number. Use -1 value, which is never be a valid ID number, for the purpose.

Note that if the optional (*row,col*) position is not given, the digit keypad will be at the default location as specified by the system initial file. The (*row,col*) position is in absolute screen coordinate.

wnd_fgeto

Open an option selection sub-window with given options being displayed in it for the operator to pick up the one desired.

int wnd_fgeto (*id, opts, atrb, nrow, deflt*)

or (feature supported after V3.1.055)

int **wnd_fgeto** (*id,mid,atrb,nrow,deft*)

- int *id*** Integer number, specifying the screen item field's ID number, where the option sub-window is to open below, provided that it is a valid one (which must be positive, and may start from zero if no title bar is used for the window). If the ID number is not valid for current window, then the sub-window will be opened just below the current cursor position.
- char *opts*[][]** Array of character strings containing the option descriptions. The number of options is determined by the number of strings passed by the array; however, a null string in the array will terminate the counting of the number.
- int *mid*** Memory text buffer handle returned by **tx_open()**. The content of the memory text buffer will be the content of the list box for user selection.
- int *atrb*** Integer number, specifying the color attribute (text foreground and background color) of the sub-window.
- int *nrow*** Optional integer number to specify the maximum number of character rows to use for the option sub-window. If this argument is omitted or is of wrong type, the default number will be 8.
- int *deft*** Optional integer value for the default option. If this argument is not present, or is of wrong type, the default will be zero. The default value will be returned if the operator presses the <ESC> key or the right most button of the pointing device to quit the sub-window.
- Return** Integer value representing the option selected, from 0 to *n-1*, where *n* is the total number of options. If a memory text buffer is used instead of the string array, it will return -1 if the memory text buffer handle is not valid.

This function is used to drop down an option selection sub-window for the operator to select one from within. A slide button will be used automatically at the right of the window if the total number of options is greater than the maximum number of character rows allowed of the sub-window. The width of the sub-window is calculated automatically by the passed option strings, while the height of it is also adjusted if the number of options is less than the maximum allowable.

Though the number of option strings is determined by the number of string (a single array of character) elements in the passed array, it is better to include a null string at the end of the string array to avoid possible porting problem in the future. Note that in standard C implementation, the argument will be passed as a pointer to an array of pointer to character string (***opts[]**), and there may be no way for the function to know the number of array elements.

For version after V3.1.055, the **wnd_fgeto()** supports the use of a memory text buffer for the content of the selection list. This feature is very useful if you have a very long list of selection, since a string array can take up at most total 65536 bytes in memory size from current release. See **tx_open()** for the creation of a memory text buffer.

Note also that the character string after '\n' and 0x01 code in each record from the memory text buffer will not appear in the selection window. This fact is very helpful to hide some related informations in the same record, recalling that a record can have at most 256 characters.

To test if this feature is present, test the presence of the **filesize()** function which is added at the same time.

wnd_fgets

Open a text entry field for the operator to enter or modify a text string.

String **wnd_fgets** (*id,npos,atrb,deft*)

- int id** Integer number, specifying the screen item field's ID number, where the entry field is to open, provided that it is a valid one (which must be positive, and may start from zero if no title bar is used for the window). If the ID number is not valid for current window, then the field will be opened at the current cursor position.
- int npos** Integer number, specifying the width of the entry field.
- int atrb** Integer number, specifying the color attribute (text foreground and background color) of the entry field.
- char deft[]** Optional default string for the entry field. If this argument is present, the text entry field will be initialized with it.
- Return** String value that the operator has just entered to the field.

This function is used to open a text entry field, with/without an initial default, for the operator to enter or modify a text string.

If a valid screen item field's ID number is supplied, then the entry field will be opened starting from the beginning of the item field. However, the TCL programmer must supply the width of the entry field explicitly, which should be the same as the one of the item field for visual consistency.

The text entry field will be opened at the current cursor position, if the first argument is not a valid item field's ID number. Use -1 value, which is never be a valid ID number, for the purpose.

Windows Specific

An additional bit flag control is added to *atrb*:

Bit 11: ON, input in password mode, i.e., the character echoed in the field will all be '*'s.

wnd_field

Set up a screen item field at current cursor position.

int **wnd_field** (*type,atrb,size*)

- int type** Integer number, specifying which type of screen item is to setup. Currently supported types are:
- 0** General text box with several kinds of border support, the size of which is specified by width and height in text column and text row

number respectively. This kind of screen item can be setup for text entry fields and screen buttons.

- 1** Square box of fixed size, used for status marking. This kind of screen item is used to indicate a certain states of options. Two kinds of marking can be done two such item in display. One is make a cross mark of 'X' in the box, and the other is to fill it up with specific solid color.
 - 2** Standard Windows Edit Box Control, valid only in Windows version. The *atrb* will specify the attribute of the edit box. See **wnd_editbox()** for details.
- else** No effects.

int atrb Integer number, bit fielded, specifying the color attribute (text foreground and background color) and border type of the screen item field. The meaning of each bit fields are given below:

Bit 0-3: Foreground color of text and border.

Bit 4-7: Background color of the field.

Bit 8-10: Type of border or style of the button. Useful values are 0 for no border, 1 for value entry, 2 for double border of value entry, 3 and 4 for screen button.

Bit 11: Not use in DOS version.

Bit 12: Active control. If it is 0, the screen field is activated immediately when the operator depresses the button, and 1, the activation must wait for the operator to release the button. This control is effective only in DOS version.

Bit 13: Repeating control. If it is 0, the activation of the screen item field is only one shot. If it is 1, then the activation will be repeated as long as the button is being depressed, provided that the active control bit is also 0. The repeating of activation starts after the button being held down at about 0.5 second. The repeating rates depends on how fast the activation is serviced. This control is effective only in DOS version.

Bit 14: Screen activation control. If it is 0, the area of the screen item field is activated (to change appearance) whenever the cursor pointer is moving across. This helps the operator to identify which of these screen item fields are currently valid for the operation. It it is 1, then there will be no such activation provoked. This control is effective only in DOS version.

int size Integer number, specifying the size of the screen item field. The low byte specifies the width of the field in units of text column, and the high byte, the height in units of text row. If the high byte is zero, then 1 text row is assumed. Note that the cursor position will be the lower left corner of the screen field.

Return Screen item field's ID number in integer, which is sequentially assigned from 0 when a graphic text window is opened. Note that the title bar also takes one screen item field. So, if the window was opened with a title bar, this screen item field's ID number will starts from 1.

This function is valid only when there is graphic text window being active. It is used to setup screen items which can be picked up by the operator via the **wnd_gbtn()** function calls. Note that what this function provides is only a declaration of a screen area, and an initial graphic layout of this area. Programmer must use other functions such as

wnd_ftext(), **wnd_fmark()**, **wnd_fgets()**, **wnd_fgeto()** and **wnd_fgetd()** together with the screen item field's ID number returned by this function to provide the operator a GUI environment.

Windows Specific

For type 0 screen item creation, TwinCAD will create different button controls based on the style specified in *atrb* bit 8-10:

- Style 0: Plan text button without frame or border, compatible with that created by the DOS version. The foreground and background colors are effective to define its appearance. When the mouse pointer is crossing, it will be reversed in color.
- Style 1: Entry box with thin frame around, compatible with that created by the DOS version. The foreground and background colors are effective to define its appearance. However, it is reversed in color only when the left mouse button is depressed.
- Style 2: Color Box button. The foreground and background colors are effective to define its appearance. Its activation follows the Windows convention. Note that if both the foreground and background colors are the same, the button text will be displayed in black with 3D effect.
- Style 3: Standard Windows Button. Its appearance is determined by the Windows system. The foreground and background colors are not effective.
- Style 4: Same as Style 3.
- Style 5: Reserved, same as Style 3.
- Style 6: Reserved, same as Style 3.
- Style 7: Reserved, same as Style 3.

The bit 12 and bit 13 are now used to specify whether the button will be moved to a new position when its parent window is resized in vertical or in horizontal direction, respectively. If the bit flag is ON, when the parent window is resized in the corresponding direction, it will be moved accordingly.

For type 1 screen item creation, TwinCAD will create a check box initially. However, depending on subsequent **wnd_fmark()** function calls, it may be changed into a radio button. See **wnd_fmark()** for further details.

To set and to change the button text, use **wnd_puts()** at the button location. TwinCAD will automatically locate the button and set the text. The text will always be centered on the button face.

If the bit 11 (bit flag value 0x800) is set, the box button will be created using the internal status color palettes instead of the GUI palettes. This features is valid only after V3.1.055.

wnd_floct

Locate cursor position to start of a specific screen item field.

void **wnd_floct** (*id*)

int id Integer number, specifying the screen item field's ID number, where the cursor position is to be located to, provided that it is a valid number (which must be positive, and may start from zero if no title bar is used for the window). If the ID number is not valid for current window, then the cursor position will not move.

wnd_flow

Draw flow line between specific screen items.

void **wnd_flow** (*from,to,type,atrb*)

int from Integer number, specifying the flow line starting screen item field's ID.

int to Integer number, specifying the flow line ending screen item field's ID.

int type Integer number, specifying the type of flow line to draw according to the rules below:

Low byte Specifying the direction code of flow line outlet from the starting item field.

High byte Specifying the direction code of flow line entering into the ending item field, where an arrow is drawn.

Valid direction codes are as below:

- 0** To the left side of the item field.
- 1** To the top side of the item field.
- 2** To the right side of the item field.
- 3** To the bottom side of the item field.

int atrb Integer number, specifying the color attribute of the flow line to draw.

Windows Specific

TwinCAD will ignore this function for the time being.

wnd_fmark

Mark a specific screen item field.

void **wnd_fmark** (*id,type,atrb*)

int id Integer number, specifying the screen item field's ID number, where to generate the mark, provided that it is a valid number (which must be positive, and may start from zero if no title bar is used for the window). If the ID number is not valid for current window, then nothing will be done.

int type Integer number, specifying what type of mark to be generated on the item field. Currently supported values are:

- 0** Generate a cross 'X' mark by drawing two diagonal lines on the item box.
- 1** Generate a solid color fill over the item box.

int atrb Integer number, bit fielded, specifying the color attribute of the mark, as described below:

- Bit 0-3:** Color value of the solid fill or line drawing.
- Bit 4-7:** Reserved.
- Bit 8-9:** Color fill/drawing operation, **00:** direct write, **01:** AND operation, **10:** OR operation, and **11:** XOR operation.
- Bit 10-14:** Reserved.
- Bit 15:** If this bit is 1, the area to mark will be shrunk in for 2 pixels. This will make the border to appear clearly. If this bit is 0, there will be no shrinking at all.

DOS Specific

There is no checking about what kind of screen item being passed to this function for the marking. It is therefore very useful of this function to generate a rectangular block of solid color.

Windows Specific

If the screen field was created with type 1 using the **wnd_field()**, this function is used to set the type and the state of the screen item. If *type* is 0, the screen item will be a check box. If *type* is 1, it will be a radio button. The value of the third argument, *state*, is no longer used to specify the color of the checker or solid filler. It is now used to specify the checking state of the check box or radio button. A zero means non-checking state, and non-zero, checking state.

However, for compatible reason, the following two values of state are reserved for special meaning:

- 0x8000** Set checking state.
- 0x80ff** Set non-checking state.

wnd_ftext

Write text to specific screen item field.

void **wnd_ftext** (*id, str, atrb*)

int id Integer number, specifying the screen item field's ID number, where the text is to write to, provided that it is a valid number (which must be positive, and may start from zero if no title bar is used for the window). If the ID number is not valid for current window, then the text will be written to the current cursor position.

char str[] String to write to the text field.

int atrb Integer, specifying the attribute of the text in display. The meaning of each bit fields are described in **wnd_puts()**.

This function is implemented by calling the *wnd_floct(id)* first and then *wnd_puts(str, atrb)* next to generate the texts. It is therefore the caller's responsibility to take care of the length of the texts to write to. There is no checking whether the text writing will

overshoot the screen item field. Programmer may use the **prints()** function to format the text string to output. Examples are given below:

```
wnd_ftext(1,prints("%8.3f",dval),0xf1);  
wnd_ftext(2,prints("%-32s",str),0x2f1);
```

wnd_gbtn

Enter GUI mode and get user button response.

int **wnd_gbtn(flag)**

int *flag*

Optional control flag, Default to 0. See below:

- Bit 0** Reserved, Should be 0.
- Bit 1** 1, Backward search of screen item, 0, forward (DOS only).
- Bit 2-11** Reserved, Should be 0.
- Bit 10** 1, Return 0xff00 if no valid input (like keyin(), Windows Only).
- Bit 11** 1, Bypass all windows messages (non-modal, Windows Only).
- Bit 12** 1, Return Any Key Press (Return CR if pick up nothing)
- Bit 13** 1, Return if Pick on Current Text Window (DOS only).
- Bit 14** 1, No check on keyboard code (like cursor key...)
- Bit 15** 1, Blink Cursor at current active cursor position (DOS only).

The flag value is passed directly to the Graphic Runtime (after V2.70) in the DOS version.

Return

Integer value of the user response via the input devices:

- 1** User has pressed <ESC>, <Ctrl/C> key.
- 2** User has pressed <END> key or the middle button of the pointing device.
- 3** User has pressed <Home> key or the right most button of the pointing device.
- 0-255** Screen item field's ID number that the user has just picked up. Note that the high byte of the returned value must be zero.
- else** Key value from the keyboard or equivalent device. The high byte is the scan code value, and low byte the equivalent ASCII code.

DOS Specific

When this function is activated, the GUI mode is entered. A cursor pointer (a shape of fingers) will appear in the screen for the operator to control the movement. The operator thus moves the pointer to pick up items of interest from the window. This function returns when the operator has made any input.

Note that the input from the digitizer not digitizing in the screen area is masked off in current version of TCL support, since there is no way for a TCL program knows how the tablet is setup.

wnd_getsize Win

Retrieve the current window's parameters

int wnd_getsize (*wnd*)

int *wnd[5]* Integer array of at least 5 elements to receive the window's parameters (top, bottom, left, right and display attribute).

Return True, if successful, False, if not.

The current active window may be resized or moved to a new position in Windows environment, and the color attribute may have been changed for a more desirable value. If it is necessary for a TCL application to keep those changes, it may call this function to retrieve the window's parameters before closing it.

wnd_group Win

Create a static group control.

int wnd_group (*row, col, width, height, atrb, text, tatr*)

int *row* Row specification of the static group control's position. See **wnd_loct()**.

int *col* Column specification of the static group control's position. See **wnd_loct()**.

int *width* Column width of the group box given in Column specification.

int *height* Row height of the group box given in Row specification.

int *atrb* Optional integer, attribute of the group box control.

char *text[]* Optional group box caption text (title).

int *tatr* Optional integer, color attribute of the group box caption text.

Return Meaningless, in current release.

wnd_handle Win

Obtain the window handle from screen item ID.

int wnd_handle (*id*)

int *id* Integer number, specifying the screen item field's ID number of which the window handle is to return. This argument is optional. If no argument is given, the current selected window's handle (In TCL system, not in Windows system) will be returned.

Return NULL, if no valid handle is found; otherwise, the window handle of the control element identified by the given ID number.

wnd_icon

Create a tile of icon at current cursor position with optional screen item field attribute setup.

int **wnd_icon** (*type, atrb, size, sld-spec*)

int type Integer number, specifying which type of tile icon to setup. Currently supported types are:

- 0** Tile of slide without screen item setup.
- 1** Tile of slide with screen item setup.
- 2** Tile of bit-map icon with screen item setup.
- else** Reserved. No effects.

int atrb Integer number, bit fielded, specifying the color attribute (text foreground and background color) and border type of the screen item field or tile. See *atrb* description of **wnd_field()**.

int size Integer number, specifying the size of the tile or screen item field. The low byte specifies the width of the field in units of text column, and the high byte, the height in units of text row. If the high byte is zero, then 1 text row is assumed. Note that the current cursor position will be the lower left corner of the tile.

char sld-spec String, specification of an icon from external file which can be

- ⇒ The filename of a slide file without any leading spaces, or
- ⇒ The filename of a slide library without any leading spaces, followed by the name of a slide in the library enclosed with a pair of parenthesis, or
- ⇒ The filename of an icon file without any leading spaces, or
- ⇒ The filename of an icon library without any leading spaces, followed by the name of an icon in the library enclosed with a pair of parenthesis.

Return Screen item field's ID number in integer if a screen item field is setup; otherwise, 0.

This function is valid only when there is graphic text window being active. See **wnd_field()** for related information.

The **TwinCAD** DOS package contains utilities to produce TCAM Icon File (ICF) and icon library (ILB). Such as the ICFFILE.OVL used in PRPLOT command to print to an icon file, the ICONLIB.EXE to group icon files into a library archive, the ICO2ICF.EXE to convert Window's icon file (ICO) to TCAM Icon file (ICF), and the BMP2ICF.EXE to convert BMP file into ICF file.

Windows Specific

For type 0 and 1, in addition to the original slide file specification, the *sld-spec[]* may now contain specification for Windows Bitmap or Icons. That is, if the specification is not a slide, then it will be checked to see if it is:

- A predefined name or ID number of an Icon or Bitmap from the system or TwinCAD's built-in resource. An ID number can be given in "#*nnn*" form, where *nnn* is the ID number of the resource.
- An external windows bitmap file (BMP/DIB/RLE).
- An external windows icon file (ICO).
- An external windows program (EXE/DLL) from which a specific icon is to extract with an index. For example, "Winhelp.exe(1)" will extract the second icon from the Winhelp.exe file.
- An ICF file or element of an ILB file.

Note that the bitmap or icon image will be stretched to fit into the box as specified by the *size* parameter, if bit 11 of *atrb* is not 1.

Type 2 is treated the same as type 1.

A bubble text information may also be included in the *slide[]* argument after the slide file specification. It must be introduced by an '!' character. For example, "WinHelp.exe(0)!Help", will take the string "Help" as the bubble text. Note that the total length of a slide file specification including its associated bubble text information can not exceeds 70 characters.

Additional bit flag control in the argument *atrb* is given below:

Bit 8-11: Type of the button border. The border size will be 0 for type 0 and type 7, 2 pixels for type 1, and 4 pixels for all the other type.

Bit 11: ON, if the *sld-spec* is a bitmap image, it will not be stretched to fit over the button face if the button face is larger then the image in both width and height. The image will be centered adjusted. OFF, the bitmap image will be stretched to fit whatever.

Note that if *type* is 2, the bitmap image will be drawn using the white color as the transparent background from the image.

wnd_itemno

Screen item control operation function.

int **wnd_itemno** (*cmd*, *itemno*)

int *cmd*

Command code, see below:

- 0** Return current active item number.
- 1** Push all active screen items to 'stack', and set the active item number to zero. All the screen items are disabled. The programmer may create new items from this point. The newly created item's ID will start from zero.
- 2** Pop the last pushed screen items back to active state. The previous items will be restored back. All the screen item that was

created after the push still in effect, but their ID numbers are incremented by the number of items restored from the stack.

- 1 Directly move the item stack top pointer to set the number of active screen item. The second argument, *itemno*, is used to specify the details.

If the *itemno* is greater than zero, then it directly set the item number to the specific value. However, if the number is greater than the current active number, then it is ignored. If the *itemno* is equal to zero, it effectively clear all the active item number.

If the *itemno* is less than zero, it is added to the current active item number, and effectively decrement the active item number (pop off). However, if the active item number after the decrement is less than zero, it will be set to zero.

Else Reserved, and return current active item number only.

int *itemno* Parameter to specify item no, optional, depending on command code.

Return The current top active item number in the window.

This function directly calls the **TCAM/Graphic Runtime** to manage the screen item stack. The **TCAM/Graphic Runtime** use a Stack of 256 cells to manage the creation, checking, deactivation and obliteration of the screen items in current release. Two stack pointers are used in the implementation, namely, a base pointer and a top pointer. An additional pointer stack of 32 cells in depth is also used to control the change of the base pointer.

When a screen item is created, it is push onto the stack top pointed by the top pointer. The distance from the top pointer to the base one is returned as its ID number. When a screen item is referenced by an ID number, the actual item is located from the stack by adding the ID number with base pointer.

The 'push or pop' of all active screen item is to push/pop the value of the base point to/from the additional pointer stack, and modify the base pointer by the top pointer or by its previous value.

Care has to be taken by the TCL programmer not to abuse these functions, although the TCL interpreter will try to maintain the system's integrity.

wnd_listbox Win

Create a standard Windows Listbox control.

int **wnd_listbox** (*row, col, size, atrb, text*)

int *row* Row specification of the listbox control's position. See **wnd_loct()**.

int *col* Column specification of the listbox control's position. See **wnd_loct()**.

int *size* Size specification of the listbox control. See **wnd_field()**.

int *atrb* Integer, attribute of the listbox control. See description text later.

char text[[]] Optional text to be placed in the list box initially.

Return Field ID.

This function is used to create a Window standard list box control. The behavior of such control shall follow the Windows specification about it. The TCL program will receive an activation from this kind of controls only when the operator has changed or made a selection from within the list box.

To obtain the current selected text from a list box, use **wnd_settext()** function.

Note also the Windows list box will allocate memory from the local heap which is very little in TwinCAD (16-bit version). So, don't put too many things in the listbox.

The effective bit flag control values for the argument *atrb* word in standard list box creation are given below:

```
#define LBS_NOTIFY          0x0001L // Always added by TCL
#define LBS_SORT            0x0002L
#define LBS_NOREDRAW       0x0004L
#define LBS_MULTIPLESEL     0x0008L
#define LBS_OWNERDRAWFIXED 0x0010L // No use
#define LBS_OWNERDRAWVARIABLE 0x0020L // No use
#define LBS_HASSTRINGS      0x0040L
#define LBS_USETABSTOPS     0x0080L
#define LBS_NOINTEGRALHEIGHT 0x0100L // No use
#define LBS_MULTICOLUMN     0x0200L
#define LBS_WANTKEYBOARDINPUT 0x0400L // Not Processed
#define LBS_EXTENDEDSEL    0x0800L
```

See Windows SDK documentation about the details of these control flag meaning.

The following are additional bit flag control values imposed by TCL:

```
#define ES_VSCROLL          0x1000 // Add Vertical scroll
bar
#define ES_HSCROLL          0x2000 // Add Horizontal scroll
bar
#define ES_BORDER           0x4000 // Draw box border
#define ES_LARGE            0x8000 // Enlarge the size by 4
pixels
```

wnd_loct

Directly locate cursor position in current active window.

long **wnd_loct** (*rowspec*, *colspec*)

int rowspec Integer with the low byte specifying the absolute text row number (text line) of the next cursor position to be, and high byte, pixel-row adjustment. The row number taken from the low byte is always unsigned (i.e., from row 0 to 255), while the pixel-row adjustment taken from the high byte is a signed value from -128 to +127. The number of pixel-rows of a text row depends on the resolution of the graphic device and fonts in use.

int colspec

Integer with the low byte specifying the absolute text column number of the next cursor position to be, and high byte, pixel-column adjustment. The column number taken from the low byte is always unsigned (i.e., from TD -0 t

	1: Move window relative to current position. The argument <i>x</i> , <i>y</i> , <i>w</i> and <i>h</i> are taken as relative values.
int x	Integer, specifying new X position in pixel unit.
int y	Integer, specifying new Y position in pixel unit.
int w	Optional integer, specifying new window width in pixel unit. Default to 0x8000.
int h	Optional integer, specifying new window height in pixel unit. Default to 0x8000.
Return	True if successful, False if not.

If the value 0x8000 is given as an argument value for the *x*, *y*, *w* or *h* argument, it specifies not to change the window size in the corresponding measurement.

wnd_puts

Directly put text string to the current active window at current text cursor position, with specific text attribute.

void wnd_puts (*str*, *atrb*)

char <i>str</i>[]	Text string to be displayed at the current cursor position in the active window.
int <i>atrb</i>	Integer, specifying the attribute of the text in display. The meaning of each bit fields are described below: <ul style="list-style-type: none">Bit 0-3: Foreground color of the text font.Bit 4-7: Background color of the text font.Bit 8: Standard font size. 0 for normal size, 1 for small size, effective only in DOS version.Bit 9: Extension font usage. 0 for no extension font usage, 1 for using active extension font, effective only in DOS version.Bit 10: Center adjustment function override. If this bit field is 1, the text will be center adjusted at the current line, despite of the current cursor position.Bit 11: Reserved.Bit 12: Double width function. If this bit field is 1, the text will be displayed in double width. However, in Windows version, each character will still take up two display position, but the font is not scaled.Bit 13: Double height function. If this bit field is 1, the text will be displayed in double height. Not supported in Windows version for the time being.Bit 14: Underline function. If this bit field is 1, the text will be displayed with underline. Not supported in Windows version for the time being.Bit 15: Disable background. If this bit field is 1, the background color at the text position will not be modified by new text background color.

The cursor position will be updated as the text is displayed.

DOS Specific

Depending on the initial setup of the **TCAM Graphic Runtime**, the type of language supported by the extension fonts are varied. If the extension font usage is specified in the text attribute, the text codes will be checked by the extension font driver first. If it found the text code or a combination of text codes are valid, it will display them in the extension font accordingly. Otherwise, it turns the codes to the built-in driver for standard font display.

Windows Specific

When a **wnd_puts()** is called to write a text to the active window at current cursor position, TwinCAD will use its starting as well as its center position to probe the window to see if there is any active control hit by the text. If there is, the text will become the caption text of that control. If the control is a button, its caption text will be totally replaced by the new text, not concatenated nor partially overwritten.

If a text is taken as a caption text of a control, the given display attribute will be ignored. That is, it can not change the display attribute of the control.

The following bit flag controls are added in *atrb*:

- Bit 8-9:** If both bits are ON, the text will be displayed with a 3D-effect if it is not caption text of any control.
- Bit 11:** ON, probe the position that is half character width to the left of the text to display for the check box created earlier. This will ensure that the check box can be found and bound with the text.
- Bit 13:** ON, use the current selected logical font to display the text. OFF, use the internal fixed font to display the text. Note that only texts in fixed font can have the terminal display effect.

wnd_rcbox Win

Create radio button or check box.

int **wnd_rcbox** (*row, col, text, atrb*)

- int row** Row specification of the button position. See **wnd_loct()**.
- int col** Column specification of the button position. See **wnd_loct()**.
- char text[]** Optional caption text of the button.
- int atrb** Optional initial attribute of the button, default 0. Effective bit flags are listed below:
 - Bit 15:** 1, if the button is a radio button, or 0, if the button is a check box.
 - else:** Reserved and should be zero.
- Return** Button ID (Field ID).

The window check box can be created by **wnd_field()** using type 1 buttons without any caption text. A caption text will be added or modified only when **wnd_puts()** is called and the check box is hit by the text (within a narrow allowance). It thus requires a lot of overhead in creating such objects in Windows environment in the usual programming manner.

To create a window radio button without using this new function directly, one must create the check box first, and then using **wnd_fmark()** function the change its type to radio button.

The following TCL function examples give the equivalent function to **wnd_rcbox()**:

```
//
//      WndCheckBox() -- Create window check box
//
int      WndCheckBox(int r,int c,char text[],int atrb)
{
    int          bID;
    wnd_loct(r,c);
    bID = wnd_field(1,0x1f0,0x102);
    wnd_rloct(0,2);
    wnd_puts(text,atrb|0x800);
    return bID;
}
//
//      WndRadioBox() -- Create window radio button
//
int      WndRadioBox(int r,int c,char text[],int atrb)
{
    int          bID;
    wnd_loct(r,c);
    bID = wnd_field(1,0x1f0,0x102);
    wnd_rloct(0,2);
    wnd_puts(text,atrb|0x800);
    wnd_fmark(bID,1,1); // Force to change state
    wnd_fmark(bID,1,0);
    return bID;
}
```

wnd_rect Win

Create a static solid color fill rectangle.

int **wnd_rect** (*size*, *atrb*, *color*)

int size Integer number, specifying the size of the screen item field. The low byte specifies the width of the field in units of text column, and the high byte, the height in units of text row. If the high byte is zero, then 1 text row is assumed. Note that the cursor position will be the lower left corner of the screen field.

int atrb Type and attribute of the rectangle. See **wnd_field()** for frame type.

long color Color of the rectangle in RGB format.

Return 1 if the creation is successful, 0 if not.

This function is supported after V3.1.055.

wnd_resize Win

Change the size of the selected window.

long **wnd_resize** (*row, col, flag*)

- int row** New row specification, or new width in pixel, depending on bit 14 of *flag*.
- int col** New column specification, or new height in pixel, depending on bit 14 of *flag*.
- int flag** Optional, default 0, specifies how the arguments are given:
- Bit 14:** ON, if the argument is given in unit of pixel in the x and y order; i.e., the first argument specifies the width value, and the second argument, the height. OFF, 0, if the argument is given in the usual row/column specification.
 - Bit 15:** ON, if the new size is given relative to current size; i.e., positive to enlarge and negative to shrink the window. OFF, if the new size is the required absolute size of the window.
 - Else** Reserved, should be zero.
- Return** Size of the windows client area in Row/Column specification if bit 14 of *flag* is 0, and in pixel unit if bit 14 is 1.

This function is used to resize the current selected window to a desired size. It is useful in hiding or revealing additional buttons or text informations from the window.

wnd_rloct

Locate relative cursor position in the current active window.

long **wnd_rloct** (*rowspec, colspec*)

- int rowspec** Integer with the low byte specifying the relative text row number (text line) of the next cursor position to be, and high byte, pixel-row adjustment. Both the low byte and high byte are signed number.
- int colno** Signed integer specifying the relative text column number of the next cursor position to be.
- Return** The previous cursor position with its row value in high word and column value in low word.

This function is used to locate cursor in relative mode, effective only when there is active graphic text window.

wnd_rolldn

Scroll down a specific portion of graphic texts.

void **wnd_rolldn** (*wnd, nline*)

- int wnd[5]** Integer array of window construct relative to the current selected window. That is, the row 0 and column 0 start from the current selected window's row 0 and column 0. For details of the window construct, see **wndopen()**'s description. Note that the portion of window it specifies must be totally within the current selected window for an expected result.

int *nline* Number of text lines to scroll down. New text lines with the background color specified by **wnd[4]** will appear at the top of the specified window portion. If this value is zero, all the graphic text portion specified by the window construct will be cleared, as if all text lines are scrolled down to vanish.

wnd_rollup

Scroll up a specific portion of graphic texts.

void **wnd_rollup** (*wnd*, *nline*)

int *wnd[5]* Integer array of window construct relative to the current selected window. That is, the row 0 and column 0 start from the current selected window's row 0 and column 0. For details of the window construct, see **wndopen()**'s description. Note that the portion of window it specifies must be totally within the current selected window for an expected result.

int *nline* Number of text lines to scroll up. New text lines with the background color specified by **wnd[4]** will appear at the bottom of the window portion. If this value is zero, all the graphic text portion specified by the window construct will be cleared, as if all text lines are scrolled up to vanish.

wnd_runit

Change the scan line number per character row unit used in the current active window.

int **wnd_runit** (*scanno*)

int *scanno* Integer number, specifying the scan line number per each character row unit. It must be in the range from 2 to 255; otherwise, the current scan line number per each character row will not change.

Return Maximum row number of the window after setting the new scan line number per character row unit.

This function will affect all subsequent character row addressing in the graphic text window, until the window is closed or a new unit number is specified. However, it will not affect the text character height. It is used to change the basic unit used in the row specification.

After version V3.02 (inclusive), you may use **wnd_runit(0)** to obtain the maximum row number of the current selected window, without affecting its basic unit of row specification.

Windows Specific

The character row distance will be reset to default for any new windows being opened or closed. This is different from the TCAM Graphic Runtime, where the row distance setting stays with the window.

wnd_select

Select a graphic text window by its ID number.

int **wnd_select** (*wid*)

int wid Graphic text window ID number obtained by **wndopen()** or from system variable tables. If this argument is zero, then the active window is reset to the current topmost window.

Return Graphic text window ID number of the selected window. If the passed argument is a valid window's ID number, this returned value should be the same as it; otherwise, it returns the current active window's ID.

This function is used to direct the graphic text window operation to a specific window that is not currently active (on top). Care must be taken in using this function! The passed argument as the graphic text window's ID number must be valid (actually obtained by the **wndopen()** or by the system variables); otherwise, the subsequent window operation may not behave as what are expected.

The system will open several graphic text windows, for the drawing, screen menu, command text area and other usage, during the system initialization. The ID numbers of these windows can be found from the system variables (see the document about the system variables). You may locate these windows and direct the text output to there, as long as the specific window has been opened.

The selected window is active in a temporary basis. Whenever a new graphic text window is opened or closed, it will be deselected automatically. The current active window is always the topmost window (the last opened one) unless being explicitly redirected by this function. To reset the selected graphic text window back to the topmost one, pass the argument by a zero ID number (though any wrong ID number may do).

wnd_settext Win

Set/Get the caption text of a window or a window control.

STRING wnd_settext (id, text)

int id Windows ID or Field ID, or Window handle.

char text[] Optional, string to set to the window or window control.

Return String of the window text read from the control, before setting the new text.

This function is useful in reading back the current display text of the screen item.

If the window control is a slide button, this function call is the only way to effectively change the slide-specification of it.

If the window control is a Windows standard listbox or editbox, you may retrieve the text informations in the following way:

STRING wnd_settext (id, which)

int id Windows ID or Field ID, or Window handle of a listbox or editbox control.

int which Integer, specifying which line of text to be retrieved from the control. For listbox control, a -1 means to retrieve the current selected one.

Return String of the text retrieved from the control.

wnd_state Win

Set/Obtain the state of a window or window control.

int wnd_state (*id, cmd, state*)

- int *id*** Windows ID or Field ID, or Window handle.
- int *cmd*** Optional command code, default 0. Currently only the following command values are supported:
- 0** Get Window State, default.
 - 1** Set Window State.
- int *state*** Optional window state value, default 0. Valid bit flags are given below:
- Bit 0:** 1, if it is disabled, 0, if it is enabled.
 - Bit 1:** 1, if it is hidden, 0, if it is visible.
 - Bit 4:** 1, if it is in checked state, 0, if it is not.
 - Bit 6:** 1, if it is in highlight state, 0, if it is not.
 - Bit 7:** 1, if it is in focus, 0, if it is not.
 - Bit 15:** 1, if it is to set button states of bit 4-7, 0, don't change them.
 - Else:** Reserved, should be zero.
- Return** The current state or last state of the specific window or window control.

wnd_static Win

Create standard Windows static control.

int wnd_static (*row, col, size, atrb, text*)

- int *row*** Row specification of the static control's position. See **wnd_loct()**.
- int *col*** Column specification of the static control's position. See **wnd_loct()**.
- int *size*** Size specification of the static control. See **wnd_field()**.
- int *atrb*** Integer, attribute of the static control. The lower byte of this argument specifies the color attribute of the static control, if applicable. If both the background and foreground colors are the same, the static control (text) will be drawn in transparent mode.
- The lower nibble of the high byte (bit 8 to bit 11) specifies the type of the static control:
- 0:** Left justified text.
 - 1:** Center justified text.
 - 2:** Right justified text.
 - 3:** Icon. Argument *text* specifies an icon (internal only).
 - 4:** Black fill rectangle (*text* ignored).

- 5:** Gray fill rectangle (*text* ignored).
- 6:** White fill rectangle (*text* ignored).
- 7:** Black Frame (*text* ignored).
- 8:** Gray Frame (*text* ignored).
- 9:** White Frame (*text* ignored).
- B:** Simple Text (No justification).
- C:** Left justified text without word wrapping.

Bit 12 to bit 15 are control flags:

- Bit 12:** Reserved, should be zero.
- Bit 13:** Reserved, should be zero.
- Bit 14:** Generate 3D effect of text.
- Bit 15:** Assign and return a field ID.

char *text*[] Optional text for the static control.

Return If bit 15 of *atrb* is ON, field ID of the static control is returned. Otherwise, it return 0 if the static control is created successfully, and -1 if it fails.

This function is used to create a Window standard static control, which can be a static text, frame, rectangle, or icon image. Different from the static text created by **wnd_puts()**, the text style used in such static control will be in Windows default which is in proportional text font.

If bit flag 15 of *atrb* is set to 1, **wnd_static()** will return a valid field ID of the static control, which can be used in **wnd_settext()** to retrieve and to modify the static text.

wnd_style Win

Set or Get windows style flag.

long **wnd_style** (*id*, *style*, *cmd*)

int *id* Windows ID or Field ID, or Window handle.

long *style* Optional style flag value to set. If this argument is given, the current style flag of the specified window will be modified by this value, based on the value of the third argument *cmd*.

int *cmd* Optional integer, specifying how the style flag will be modified by the argument *style*:

- 0** Default, set the style flag directly.
- 1** OR with the style flag (set specific flag values).
- 2** XOR with the style flag (toggle specific flag values).
- 3** NOT the style flag (remove specific flag values).

Return The last window style flag value before modifications.

This function call SetWindowLong() and GetWindowLong() API to access the windows Style long word. See Windows SDK for more informations on windows/control styles.

wnd_textbox Win

Create a TwinCAD private text box control.

int wnd_textbox (row, col, size, atrb, tid)

- int row** Row specification of the text box control's position. See **wnd_loct()**.
- int col** Column specification of the text box control's position. See **wnd_loct()**.
- int size** Size specification of the text box control. See **wnd_field()**.
- int atrb** Integer, attribute of the text box control.
- int tid** Integer ID of the memory buffer handle returned by **tx_open()**.
- Return** Handle of the text box.

This function is used to create a TwinCAD private text edit box control which is intended for special applications. The detail specification of such text edit box is not given here and may be changed in future release. The use of this text edit box in user application is therefore not recommended.

wnd_vport

Set current graphic text window as a drawing view port.

int wnd_vport (cmd, ...)

- int cmd** Command word. The actual arguments required depend on the command word. See these prototypes given below.

int wnd_vport (0, pWmin, pWmax, clr)

- POINT pWmin** Minimum drawing mapping extent point.
- POINT pWmax** Maximum drawing mapping extent point.
- int clr** Optional, 1 to clear the view port, 0 not to. Default 1.
- Return** True, if ok, False, if not.

This function is used to setup the current active graphic text window into a drawing viewport. All the subsequent **plotent()** function calls will be directed to plot on this new viewport, instead of the system's drawing area. It is effective only when there is a user graphic text window active on the screen.

The background color of the viewport will be the same as the background color of the graphic text window. Note that if the optional third argument is not given, this function will clear the viewport after the setting up the required mapping information for the subsequent plottings.

The user drawing viewport setup by this function call will be deactivated when

- A new **wnd_vport(0,...)** function is called for new viewport setup, or
- Any graphic text window is closed (not necessarily the one containing the viewport), or
- An explicit **wnd_vport(-1)** function call is issued.

Note that for the subsequent plotting activities, the following drawing capabilities are disabled if a user viewport is being active:

- Saving the plotting vector to the display list is disabled, so as to protect the main drawing display list from being corrupted.
- Erasure of plotting vector from the display list is also disabled.

Also note that the **redraw()** function will be affected by this **wnd_vport()** function call, since it will refresh the display list in the current active viewport. When a user viewport is being active, the **command()** and **regen()** functions should not be used.

int **wnd_vport (1,xdir,ydir,zdir,zh,mark)**

double xdir	X-component of the 3-D view point.
double ydir	Y-component of the 3-D view point.
double zdir	Z-component of the 3-D view point.
double zh	Optional height of the view point center. Default 0.
int mark	Optional tripod mark specifier. True if a tripod mark will be drawn automatically, and False if not. Default is False.
Return	Meaningless.

This function call is used to switch the viewport from a 2-D viewport to a 3-D viewport. The view center will be setup in the middle of the 2-D viewport.

int **wnd_vport (2,sldspec)**

char sldspec[]	String containing a slide file specification to show on the current graphic text window.
Return	0 if the slide is shown successfully; otherwise, it indicates error.

This function call is used to show a slide on the current active graphic text window. If there is an active viewport, the viewport will be canceled. Note that, the slide will be drawn on the current active window or sub-window, not the current active viewport.

Windows Specific

In addition to the original slide file specification, the *sldfile[]* may now contain specification for Windows Bitmap or Icons. However, the bitmap or icon image will NOT be stretched to fit into the current viewport (window). See **wnd_icon()** for more informations.

void **wnd_vport (-1)**

This function call is used to deactivate the user drawing viewport setup previously by the **wnd_vport(0,...)** function call.

void **wnd_vport (-2)**

This function call is used to clear the current active viewport provided that there is an active user viewport.

int **wnd_vport (3, id, sldspec)**

int id Integer number, specifying the screen item field's ID number of a slide button created by **wnd_icon()**.

char sldspec[] String containing a slide file specification to show on the slide button face. See **wnd_icon()** for details.

Return 0 if the slide is shown successfully; otherwise, it indicates error.

This function call is used to change the button face content of a slide button. This is a new function support after TwinCAD V3.1.052.

int **wnd_vport (4, id, pWmin, pWmax, clr)**

int id Integer number, specifying the screen item field's ID number of a slide button created by **wnd_icon()**.

POINT pWmin Minimum drawing mapping extent point.

POINT pWmax Maximum drawing mapping extent point.

int clr Optional, 1 to clear the view port, 0 not to. Default 1.

Return True, if ok, False, if not.

This function call works the same as **wnd_vport(0,...)**, except that the view port is created on the slide button face. This is a new function support after TwinCAD V3.1.052.

wnd_vscroll

Create and control a vertical scroll bar.

int **wnd_vscroll (cmd, id, atrb, total, cpos)**

int cmd Command word. Valid commands are given below:

cmd = 0 Create a new scroll bar to the right of current window.

cmd = 1 Re-initialize/Update scroll bar position.

cmd = 2 Read the scroll bar activation.

int id Base ID of the GUI object (vertical scroll bar). This argument is meaningless when the **cmd** is 0.

int <i>atrb</i>	Optional, display attribute of the vertical scroll bar, default to 0x1430. This argument is meaningless when the cmd is 2. The meaning of the bit value in this argument is given below: Bit 0-3: Border and arrow color. Bit 4-7: Solid color, if not zero, for the button (bar). Bit 8-11: Type of the border. Bit 12-15: Slot Color (in the Bar section).
int <i>total</i>	Optional, total division on the scrolling range, default to 1000. This argument is meaningless when the cmd is 2.
int <i>cpos</i>	Optional, current position of the scroll bar, default to 0. This argument is meaningless when the cmd is 2.
Return	Depends on command word, as described below: cmd=0 The base ID number assigned to the newly created scroll bar. cmd=1 The same base ID number of the scroll bar. cmd=2 The scroll bar activation codes: -2 Upper portion of scroll bar activated. Usually a [PgUp] function is expected. -1 Lower portion of scroll bar activated. Usually a [PgDn] function is expected. 0 Nothing happens, no activation since the last read yet. Else Dragged position of the scroll bar plus 1. This means that the scroll bar has been dragged to a new position.

This function provides the construction of a vertical scroll bar in the current window and two basic controls of it. One is to set the scroll bar position, and the other is to read back its activation status.

To employ a scroll bar in a graphic text window, the application must first call **wnd_select()** to set it as the current window, and then call this function to create it. A scroll bar in the GUI will occupy three screen items, and a base ID number of these screen items will be returned at its creation. And thus, the three screen items are:

- Base ID: The upper arrow button.
- Base ID+1: The lower arrow button.
- Base ID+2: The scroll bar activation.

The application must also setup the division count in the scroll bar section, as well as an initial scroll bar position, via the same function call. During the GUI operation (calls to **wnd_gbtn()**), if the screen item's ID for the scroll bar activation is returned, the

application must call this function again to read its activation status and process it accordingly.

Note that the application must call this function explicitly to set the new scroll bar position. The Graphic Runtime will not update the bar position automatically upon the user activation.

See Part 5 for an example TCL program showing the use of vertical scroll bar in an application.

wnclose

Close current active text window.

void wnclose()

This function is used to close the current active window. The last unclosed graphic text window will become the current active window, if there is.

wndopen

Open a graphic text window for graphic text display.

int wndopen (*wnd*, *title*, *atrb*, *flag*)

int *wnd*[5] Integer array of window construct. See later paragraphs for details description.

char *title*[] Optional string argument, specifying to create the title bar display in the newly open window. The content of this text string will be displayed in the title bar. If this argument is missing or is of wrong type, the title bar will not be created.

int *atrb* Optional integer argument, specifying the title bar display attribute, effective only in DOS version and when *title*[] is also specified. The attribute value controls the text display's foreground and background color in its low byte value, while the high byte value controls the type and frame color of the title bar.

In Windows version, the following bit flags are effective to control the windows behavior:

Bit 12: ON to disable the vertical resizing of the graphic text window.

Bit 13: ON to disable the horizontal resizing of the graphic text window.

Bit 14: ON to disable the <ESC> key to quit of the graphic text window.

int *flag* Optional control flag, effective only in Windows version. Currently supported bit flag values are given below:

Bit 0: ON, if the *wnd*[] is given relative to the current topmost main window or the current selected window, depending on bit 2 and bit 3, effective only when bit 1 is OFF.

Bit 1: ON, if it should be created as a child window (sub-window) to the current topmost main window. This bit flag override bit 0, and is effective only when there is no caption text.

generate the Gradient fill over the window background or not. If it is ON, bit 8-10 specifies the type of the Gradient fill.

Bit 12-14: Window frame display type number in DOS version. The actual frame type display depends on the version of **TCAM Graphic Runtime**.

Bit 15: In DOS version, set this bit ON to disable the window frame display. It is ineffective in Windows version.

Note that when the TCL program exits, all opened windows will be closed automatically. However, it is a good practice to close all the opened windows explicitly before the program exits.

Windows Specific

The Graphic Text Windows opened by **wndopen()** still can have specified background and foreground colors, but the type of the window frame is now fixed with the system platform. However, if the system platform is Windows 3.1, TwinCAD will take over to draw the window frame in 3D effect.

The title bar (or caption bar in Windows term) is now displayed in Windows system default color, no longer in color specified by the third argument. However, additional controls of the window behavior have been added using this argument. See the argument description.

If the title string is given, the window will have a caption bar, and the user may drag the window around by picking at the caption bar. If the title string is not given or given by 0 or null string, the window will be created without a caption bar, yet the user may still drag the window around by picking at the inside of the window (client area in Windows term).

If bit 11 of *wnd[4]* is 1, it specifies to apply a Gradient fill of color for the window background and the bit 8 to bit 10 of *wnd[4]* specifies the Gradient fill type:

0	From top to bottom.
1	From bottom to top
2	from left to right
3	from right to left
4	from top left to lower right
5	from top right to lower left
6	from bottom left to upper right
7	from bottom right to upper left

The initial color for the Gradient fill will be the basic color specified for the window background and will be gradually decreased in the specified direction to dark.

wndopensub

Create a graphic text sub-window on current topmost window.

int **wndopensub** (*wnd*, *ctrl*)

int *wnd[5]* Integer array of window construct relative to the main window. That is, the row 0 and column 0 start from the main window's row 0 and column 0. For details of the window construct, see **wndopen()**'s description.

int *ctrl* Optional control flag, default 0. Currently, only bit 0 is supported, all else bits should be zero for future compatibility. If bit 0 is 1, the sub-window will be opened as a screen item (that is, it is a pickable window), effective in DOS version only.

Return Graphic text window ID number.

This function is used to create a graphic text sub-window on the current topmost main graphic text window. Once it is created, it will become the current selected graphic text window for the screen output. All subsequent cursor addressing and text output are relative to this sub-window. You may use the **wnd_select()** to switch between the main window and the sub-window.

Note the sub-window will be removed automatically when its containing main window is closed.

Windows Specific

The window will be opened as a pure child window to the current selected window. The bit 0 of the second argument, *ctrl*, is not valid for the time being to specify that the window is a pickable window (acting like a button). However, additional bit flag values are added in Windows version:

- Bit 6** Open the sub-window (child) with a customized dialog frame in the style as specified by the *wnd[]* attribute word.
- Bit 7** Open the sub-window (child) relative to the current selected window and becoming to a child window to it.

wndtitle

Set up or change the content of a window's title bar.

void **wndtitle** (*title,atrb*)

char *title[]* String argument, specifying to create the title bar display in the newly open window. The content of this text string will be displayed in the title bar.

int *atrb* Integer argument, specifying the title bar display attribute, of which the value controls the text display's foreground and background color in its low byte value, while the high byte value controls the type and frame color of the title bar. This argument is effective only in DOS version.

If the graphic text window already has a title bar in display, this function will modify the existing title bar; otherwise, it will create one.

wto_ecs

Transform a 3D point from WCS to ECS.

int **wto_ecs** (*Pw,Zw,pe,ze*)

POINT <i>Pw</i>	X/Y component of the 3D point in WCS.
double <i>Zw</i>	Z component of the 3D point in WCS.
POINT <i>*pe</i>	X/Y component returned in ECS, after transformation.
double <i>*ze</i>	Z-component returned in ECS, after transformation.
Return	1, if the transformation is OK, 0, if it fails due to ECS is ineffective.

The ECS must be setup by **set_ecs()** before calling this functions. See also **ecs_org()**, **eto_wcs()** functions for related informations.

zroot2

Find the complex zeros of a quadratic polynomial.

int **zroot2** (*a,b,c,zp*)

double *a,b,c* Coefficients for polynomial: $a \cdot X^2 + b \cdot X + c = 0$.

POINT *zp[2]* Complex zeros to return.

Return Number of zeros on Real axis.

zroot3

Find the complex zeros of a cubic polynomial.

int **zroot3** (*a,b,c,d, zp*)

double *a,b,c,d* Coefficients for polynomial: $a \cdot X^3 + b \cdot X^2 + c \cdot X + d = 0$.

POINT *zp[3]* Complex zeros to return.

Return Number of zeros on Real axis.

zroot4

Find the complex zeros of a polynomial of order 4.

int **zroot4** (*a,b,c,d,e,zp*)

double *a,b,c,d,e* Coefficients for polynomial: $a \cdot X^4 + b \cdot X^3 + c \cdot X^2 + d \cdot X + e = 0$.

POINT *zp[4]* Complex zeros to return.

Return Number of zeros on Real axis.

PART 4

Hints on TCL Programming

This part of the document is a collection of the TCL programming hints presented in an arbitrary order.

- Careful Use of the Command() Function
- Updating System Status Display
- Sieving Entities From a Given Selection List
- Function that Returns an SLIST data
- Union, Intersection and Subtraction of Selections Lists
- Modification of Polyline Segments
- Changing the Number of Polyline Segments
- Controlling the Joining of Polyline Segments
- Inserting a Specific Symbol
- Accessing the Drawing Block Defining Entities
- The Attribute Tags of a Block Instance
- Don't Erase Entities That Define a New Block
- Making Good Use of the Geometry Calculation Functions
- Collecting Entities Created by Command()
- Use TCL Functions Instead of the DOS Command
- Shelling an External Program
- Suppressing System Messages
- Using the Format Specifier "%@0&f" in printf()
- Passing Argument Values to an External TCL Application
- Returning a String from a TCL Function
- Determining a Unique Name for Temporary File
- Using Successive sscanf() to Parse Data String of Variable Format
- Sorting Multiple Index-Related Data Arrays
- The Inside-test of a Point within a Polyline

Careful Use of the Command() Function

The use the **command()** to execute a predefined command script is a very common practice in TCL programming. However, the TCL programmers must be aware of the fact that the result of its execution may depend greatly on the current system operating statuses.

For example, the current running mode of the object snapping (**OSNAP**) may affect the system in interpreting the data point read from the script. And thus, the function call, **command("Pan P1 P2")**, may fail if the current **OSNAP** mode is not NONE, since the data

point P1 and P2 will be interpreted as screen positions to snap points on objects, not the data points themselves.

It is therefore important for the TCL programmer to preset these affecting system operating statuses to the presumed states before using the **command()**. An example is given below:

```
int    osmode;
osmode = @OSMODE;
@OSMODE = 0;
command("PAN p1 p2");
@OSMODE = osmode;
```

Updating System Status Display

You may change the current entity layer, color, and other system statuses by directly setting appropriate values to their corresponding system variables. However, altering these system variables does cause the system to acknowledge the change of these variables automatically by updating the display of the status line. Nevertheless, if necessary, you may issue the function call, **command("^O^O^M")**, to force the system to update the status display.

Sieving Entities From a Given Selection List

A simple way to sieve off specific entities from a given entity selection list is to take the advantage of the entity selection masking ability of the system. For example, the following codes will obtain a new selection list (SL1) containing only Circles from a given selection list (SL):

```
SLIST SL, SL1;
...
@SELMASK = ~FCIRCLE;    // ~0x10, TCLCAD.H
@SNAPFLAG | = 0x2000;   // Enable Selection Masking
command("SELECT \@SL ^M");
SL1 = getesel(-1);
...

```

Function that Returns an SLIST data

It is not possible to return an SLIST data directly from a TCL written function. A bad programming example is listed below:

```
SLIST badsel(){ return getesel(); }
```

The function *badsel()* tries to return an SLIST data, but the caller will always receive an invalid one. This is because the returning of the *badsel()* has de-allocated the data memory allocated by the *getesel()* function call!

If you have to design a TCL function that returns an SLIST data, you have to design it in this way: Let the caller allocate the data memory via **ent_alloc()** first, and pass it to your function, and your function has to copy the returning SLIST data into it. An example is given below:

```
int    selobj(SLIST rset)
{
    SLIST lset;
    lset = getesel("\nSelect Object: ");
    memmove(lset,rset);
    return ent_count(lset);
}
```

Of course, the caller must allocate an SLIST data of enough size to pass to this function.

In fact, a more simple and efficient way is to take the advantage of the Previous Selection Set of the system. The function may use the **command("SELECT ...")** and let the returning selection list be memorized in the system's Previous Selection Set. And then, the caller may use **getesel(-1)** function call to obtain the selection list, after returning from the function.

Union, Intersection and Subtraction of Selections Lists

It is sometimes desirable to obtain the union or the intersection of two given selection lists, or to subtract one selection list from another. This can be done in a very simple and efficient way: Just use the **command("SELECT ...")** function calls.

For example, to obtain an union set Sc from Sa and Sb, ie. Sc=Sa+Sb, use the program fragment:

```
command("SELECT A \@Sa \@Sb ^M");  
Sc = getesel(-1);
```

which setups Add mode and supplies the two selection lists to the SELECT command. The intersection set Sc between Sa and Sb can also be obtained by the program fragment:

```
command("SELECT A \@Sa \@Sb T \@Sa \@Sb ^M");  
Sc = getesel(-1);
```

which adds Sa and Sb (creates an union) first and then XOR (Exclusive OR) with the Sa and Sb again. The net result is their intersection.

To obtain a new Sc by subtracting Sb from Sa, use the program fragment:

```
command("SELECT A \@Sa R \@Sb ^M");  
Sc = getesel(-1);
```

That's simple, isn't it. Of course, you have to disable the entity selection masking function before doing these operations.

Modification of Polyline Segments

The system uses a double linked list to handle the Polyline structure. You may use **ent_pnext()** and **ent_plast()** to travel each segment along this linked list of the polyline. If you have to modify the geometry of a polyline segment, you have to maintain the continuity of the polyline passing through the segment, so as to maintain the consistency of the polyline definition. This means, after the modification, these segments belonging to the same polyline must joint together geometrically. Otherwise, the user may observe a strange polyline.

So, if necessary, use **command("QCHANGE ...")** to change the vertex of a polyline (which modifies two segments simultaneously).

Changing the Number of Polyline Segments

If you are going to change the geometry of a polyline by adding new segments (such as to create a notch on a line segment), a simple way is to use the QBREAK command. The QBREAK command will break one polyline segments into two segments, while maintaining the joining relationship. You can use **ent_pnext()** to obtain the newly added segment. You may then modify the new created vertex point to the point desired, using the QCHANGE command.

The FILLET and CHAMFER command can also be used extensively to add new arc or line segments in a polyline. They can also be used to remove a part of the polyline, since if they are applied to two non-adjacent segments, those segments in between will be removed.

However, if the change of the polyline is quite a lot, you may chose to explode it, make the change, and then re-join it again. Of course, you have to collect these polyline segments via **ent_pnext()** and store their entity handle to an SLIST data first before exploding the polyline. Otherwise, after the explosion, there is no simple way to re-collect them again.

Controlling the Joining of Polyline Segments

If you are going to create a polyline by joining a given set of entities (a selection list), you may use the AUTOJOIN or the PJOIN command, depending on the purpose. However, the PJOIN command will ask for an indication to make a choice when a branch is encountered in the joining. If you should know that there is a branch in advance, and you want the joining be done automatically in the way you wish, it is recommended to separate the entities at the branch into different selection lists and issue the PJOIN twice (or more).

Inserting a Specific Symbol

If you are going to insert a specific symbol into the drawing, you have to make sure that the symbol has been loaded in the system before issuing the SYMBOL command. This is because if the specific symbol is not loaded, the SYMBOL command will try to find it from the current active symbol library only. And if it can not be found there, the system will enter the user interface for a symbol library selection. This may cause trouble to the user of the application.

An example program fragment to insert a symbol is given below:

```
if ( symbol("VAL402W") )
    command("SYMBOL VAL402W p1 O 30 W 0.5 S 10");
else
    printf("\nSymbol 'VAL402W' not found.");
```

The TCL function **symbol()** is used to make sure that the specific symbol is loaded in the system before issuing the SYMBOL command. Note that the **symbol()** will automatically search through the directory paths specified in TCADPATH environment variable for all the symbol libraries to find and to load the specific symbol, if it has not yet been loaded into the system.

Accessing the Drawing Block Defining Entities

To access the defining entities of a drawing block, you have to locate the Block Control Entity first by the **eblock()** function call. Since a block entity is referenced by its name, the **eblock()** function accepts a block name as the only parameter.

The following program fragment shows the way how the defining entities of a block are accessed:

```
e1=gete();          // Assume e1 -> block instance
e2=eblock(ent_read(e1).insert.name);
                  // Obtain the Block entity by e1
for(e3=ent_bnext(e2);ent_ok(e3);e3=ent_bnext(e3))
    ...           // e3 -> each of the block element
}
```

Note that the geometry of these defining entities are relative to the block's insert base point, and the actual picture of the block instance is a transformation result of these entities under the control of the block instance's control data.

The Attribute Tags of a Block Instance

When a block instance is created by the INSERT command, only those variable attribute tags will be created to tag after the block instance. The system will not copy constant attribute tags to the block instance's local tag list. So, to access these constant attribute tag of a block instance, you will have to travel its original block entity for them.

Likewise, since the effective variable attribute tags to a block instance are those tagged after it, not these from the original block definition, changing the content of these attribute tag from the original block definition does not affect those in the local tag list.

To travel the local tag list, use `ent_tnext()` function call.

Don't Erase Entities That Define a New Block

You may define a new block with a given selection list via the `command("BLOCK..")` function call. However, don't erase the selection list afterward. The entity handles in the selection list are still effectively pointing to those entities that have just become a part of a block definition. If you erase them, you are erasing them from the block definition!

Making Good Use of the Geometry Calculation Functions

The TCL runtime library is rich of geometry calculation supports. Please look for them from the `V()`, `P()`, `L()`, `A()` and `C()`, if you do not find a suspicious function name from the table of content.

For example, to calculate the distance between two points, P1 and P2, there are many ways:

```
dist=P1,P2;
dist=V(P1,P2);
dist=ent_len(L(P1,P2));
dist=abs(P1-P2);
```

You don't have to write your own function to calculate the square root of the sum of the delta-x square and the delta-y square! If you want your program be more readable, you may define the macro function:

```
#define pp_dist(p1,p2) V(p1,p2)
```

and use it in your program.

You can also utilize the vector calculation (also complex number) to do the geometry calculation. For example, to calculate the unit vector from P1 to P2, you can write:

```
P3=(P2-P1)/abs(P2-P1);
```

and to rotate this unit vector by an angle of 30 degree counter-clockwise, you can write:

```
P3*=P(1,30A);
```

where the scale factor is set to 1 for a pure rotation. Of course, if you are going to rotate a point P1 about a given center P2, you have to translate the point first, as the example given below:

```
P3 = (P2-P1)*P(1,30A)+P2
```

If you don't like the post tagging character 'A', you may write the rotation vector in the form as `P(cos(30),sin(30))`. You may define the following two macros and include them in your `geofnc.h`:

```
#define rotate(p1,angle) (p1)*p(1,(angle)A)
#define rotatepc(p1,angle,pc) rotate(p1-pc)+pc
```

Another example of using the unit vector calculation is to calculate the point located at a given distance from P1 in the direction toward P2 with the expression:

```
P3=dist*(P2-P1)/abs(P2-P1) + P1;
```

In fact, the same purpose can be achieved simply by the expression

```
P3=P(L(P1,P2),dist);
```

that utilizes the **P()** again.

There are a lot of such examples for you to find out.

Collecting Entities Created by Command()

You can easily obtain the handle of the entity created by **command("INSERT...")** function call via the **ent_last()** function called immediately thereafter. This is because new entities are always appended to the end of the drawing database and you know that the INSERT command will create at most one entity at a time, provided that the insertion is successful.

However, if you try to explode that block instance with the **command("EXPLODE ...")** function call, you may not know in advance how many entities will be created. In such a case, if you want to collect these newly exploded entities, you should obtain the last entity from the drawing database first before the explosion, and then use **getesel(ent)** function call to collect them, as the program fragment shown below:

```
e1=ent_last();           // The Last entity
command("EXPLODE ..."); // Do the explosion
slist=getesel(ent_next(e1)); // Collect them!
```

The **ent_next(e1)** function is called to advance the entity handle by one, since **e1** is not to be included in the list.

This technique works for any **command()** function calls that may produce new entities. However, the explosion of a polyline does not produce any new entities. You will have to use the **ent_pnext()** to travel and collect each of the element of the polyline before exploding it.

Use TCL Functions Instead of the DOS Command

If you want to erase, rename or copy a single file, it is better to use the **remove()**, **rename()** and **copyfile()** functions than to issue the **command("DOS ...")** function call. This is because that using these TCL functions are faster in execution and the possible error condition can be returned and checked.

Shelling an External Program

If you want to shell an external program that needs only a small amount of memory (below 200 KB) and does not use the screen for I/O, you may use the **exec()** function to do the job in a much faster way. Otherwise, you should use **command("DOS ...")** function call to shell it.

The system will release almost all the real DOS memory (below 640KB) it has allocated and exit the graphic mode, to shell the external program under the DOS command. But, if there is an XMS memory support in the machine, the system will use up all the available XMS memory and will not release them automatically at shelling the program.

You may, however, explicitly tell the system to release some (at least half) of the allocated XMS memory before shelling the specific program by adding a special code '!' (followed by a space) before the program name, as the example given below:

```
command("DOS ! program...");
```

The only drawback for this is that the system will need to regenerate the drawing afterward (to rebuild the display list).

If you need to shell successive programs in a sequence, using the DOS command, it is recommended to create a DOS batch file first and then shell the batch file instead. This will save a lot of system swapping time.

Suppressing System Messages

To suppress the system messages during the execution of the **command()** function calls, you should turn the @CMDECHO OFF by setting it to zero. However, turning off @CMDECHO does not necessary mean that the system will not echo any input nor to issue any message to the command area. For example, when the next code to be read from the script is a '/' code, which is used to switch the system to read an effective input from the real user interface, the message suppression will be temporarily disabled until the service of the '/' code is completed.

In fact, whenever the script is not active (due to its termination or being temporarily switched-off), the message suppression will always be disabled. Most of the frustrating cases of unsuppressed system messages are resulted from this fact. For example, the expression

```
command("SELECT c p1 p2 ^M");
```

is issued to select objects by window crossing with two given points. As the last '^M' code is issued to terminate the SELECT operations, the script ends. And thus, the last message generated by SELECT command to report the state of the selection can not be suppressed due to no script being active.

To avoid this problem, you should add one or more spaces or '^M' codes to the end of the script, so that the script will not end when the SELECT command terminates and the message is thus suppressed. These additional spaces or '^M' codes will not cause any problem, since they are ineffective to the **command()** and will not cause any repetition of commands.

Using the Format Specifier "%@0&f" in printf()

The format specifier "%@0&f" is not a standard format specifier and is therefore not formally documented in the **printf()** function. However, it is actually present and used by the system internally. It is used to output a floating point value (double) in a format controlled by the system variable @DISP_POSNO according to the following rules:

- If @DISP_POSNO is zero, the output will be rounded to 3 decimal positions after the decimal point. The trailing '0' will also be removed. This is the system default.
- If @DISP_POSNO is greater than zero, it specifies the number of decimal positions after the decimal point to round the value and to output, but the trailing '0' will be retained.
- If @DISP_POSNO is negative, then its absolute value also specifies the number of decimal positions after the decimal point to round the value and to output, but the trailing '0' will be removed.

Note that the leading space of the conversion result will always be removed. Care should be taken in setting the @DISP_POSNO variable, since it will affect the system display of the coordinates, length and angle values. If you must take advantage of this format specifier, be sure to preserve the original setting value of @DISP_POSNO, and restore it back after use.

Passing Argument Values to an External TCL Application

An active TCL application may invoke another external TCL program as a child application via the **command("RUN ...")** function call. However, there is no formal way to pass argument values to the child program, nor to obtain its execution result via a formal return statement. Nevertheless, you may take advantage of the system registers and devise your own way to pass argument values and to retrieve the returning results. If the use of the system registers is not enough for your application (e.g. passing a large array of data), you may resort to the use of an external file to solve the problem.

Returning a String from a TCL Function

If you are going to write a function returning a character string, all you have to do is NOT to declare the function with any data type and directly return a string type data within that function, like the example below:

```
readstr()
{
    char    input[80];
    input = gets("\nString: ",40);
    return input;    // Not work in C/C++
}
```

This is convenient, but not compatible with Standard C. The incompatibility lies not only in the function type declaration but also in the way the data returned.

In Standard C, a string is a one-dimensional array of character type data, of which the value can not be returned formally. It must be returned by a pointer pointing to the array. But, in TCL, the one-dimensional character array can be taken as a single type of data called STRING and therefore, its value can be formally returned by the return statement.

Note that the formal way to return an array data in TCL is through the formal argument.

Determining a Unique Name for Temporary File

The TCL runtime library does not provide a function to return a unique filename for creating temporary files. However, you may use the following example program fragment to determine a unique name:

```
long count;
int  cdate[4];
char name[12];
delay(1); // make sure it won't repeat for successive calls
date(cdate);
count = (cdate[0]%3)*0x22DDD200L+
        (cdate[1]*31+cdate[2])*1572480L+tick();
name = prints("%08lx",count);
```

The name will never repeat unless the program fragment had been executed three years ago at the same timer tick! Of course, you would better check it before use it.

Using Successive sscanf() to Parse Data String of Variable Format

The return value of sscanf() indicates the number of data items that have been successfully read from the input string. This is not compatible with the Standard C/C++ library function, yet is more useful in TCL programming.

For example, to remove the leading space from the input string and test if the string is still containing data to read, use the program fragment:

```
if ( !sscanf(buf, "%256c",buf) )  
    /* No more data in the buffer to read */  
}
```

To determine whether the input string starts with a specific keyword, use the program fragment below to test and bypass the keyword from the it:

```
if ( sscanf(buf, " DRAW %256c",buf) 0 )  
    /* The keyword 'DRAW' is determined and removed */  
}
```

With such a technique, you may easily implement your own variable data format parser for your applications.

Sorting Multiple Index-Related Data Arrays

The **sort()** function will sort an array of integers, longs, doubles, character strings, POINTs, CIRCLEs and Entity handles in various ways. You can easily sort out such a single data array by passing it to the **sort()** function. However, to sort out multiple data arrays that are to

In fact, the TCL runtime library after Version 3.0/R3d has expended the **sort()** to return directly the change of ordering after sorting (see **sort()** function). You may use the following codes to detect whether your version of program has this support:

```
if ( sort(array,0) )
    /* Yes, the feature is supported */
}
```

The runtime library also included as new **setdata()** that can be readily used for the job. The sorting of the multiple index-related data arrays can then be written as the program fragment below:

```
int nIndex[nCount];
if ( sort(array,0) )
    sort(Key,nCount,0,nIndex);
    setdata(array1,array1,nIndex);
    ...
    setdata(arrayn,arrayn,nIndex);
}
else sortdata(nCount); // Old sorting routine
```

The Inside-test of a Point within a Polyline

There is no direct function call to support the inside-test of a point within a given polyline; however, such an inside-test functionality can be easily implemented via **pe_dist()** and **ent_area()** functions. An example and handy routine is given below:

```
int PtInPoly(POINT pt, ENTITY ePoly)
{
    double area, dist;
    dist = pe_dist(pt,ePoly);
    area = ent_area(ePoly);
    if ( (area>0) && (dist<=0) ) return 1; // CCW && Left
    if ( (area<0) && (dist>=0) ) return 1; // CW && Right
    return 0;
}
```

The above function will return 1 if the given point lies inside the given polyline, and will return 0 if it does not. The implementation is based on the fact that if the orientation of the polyline is CCW, a point will be inside to the polyline if and only if it is always to the left of the polyline. Likewise, if the polyline is CW, the inside points must be always to the right of the polyline.

The inside-test can also be implemented using **ent_intsc()** function calls by counting the number of intersection points of the polyline with a line-ray starting from that given point out to a far position.

PART 5

Example TCL Programs

CDA.TCL

The source program listed below is an example for building a command which makes chamfer over two given line segments. The chamfer is specified by a distance and a deflection angle from the first line segment. The programming technique applied here is quite straightforward. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```

/*****
Program:      CDA.TCL
Author:       Kang, Hsin-Min
Purpose:      Demonstration of TCL programming
Effect:       Build CDA Command to chamfer two lines with given a distance
              and deflection angle from the first line.
*****/

#include <tclcad.h>
#include <geofnc.h>

#define nearp1(p1,p2,po)  (v(p1,po)<=v(p2,po))
#define isccw(p1,p2,p3)  (ent_area(p1,p2,p3)>=0)

COMMAND Cda()
{
    double    dist;    // Distance from intersection point
    double    angle;   // Angle deflection from first line
    double    absang;  // Absolute angle direction of line
    LINE      ln;      // Temporary line data loaded
    POINT     pint;    // Intersection point

    // Obtain basic requirements

    dist = getv("\nEnter distance value: ");
    angle = abs(getv("\nEnter angle value (45): ",45));
    e1 = gete("\nSelect first line segment: ",FLINE);
    e2 = gete("\nSelect second line segment: ",FLINE);

    // Get Intersection point of two lines

    if ( !ent_intsc(e1,e2,&pint,1) ) {
        printf("\nNo intersection point!");
        exit(0);
    }

    ln = e1;          // Load first line data implicitly

    // Calculate chamfer point at both lines

    p1 = L(pint,P(e1.pick,ln)),dist;
    absang = ppangle(p1,pint); // From chamfer point to
    intersection
    if ( isccw(e2.pick,p1,pint) )
        absang += angle;      // CCW chamfer
    else
        absang -= angle;      // CW chamfer

    ln = p1,dist,(absang)A;    // Get chamfer line

    if ( !ent_intsc(ln,e2,&p2,1) ) {
        printf("\nCan't have chamfer!");
    }
}

```

```

        exit(0);
    }
    newent(L(p1,p2));           // Create chamfer line
    updent(e1,p1);             // Trim first line
    updent(e2,p2);             // Trim second line
}
/*

```

Note that if the new data point does not lie within the span of the line, then the part that dose not contain the pick point will be trimmed off by the following function. This criterion has a little defect in view of chamfer operation. Readers are encouraged to fix it by considering the case when the pick point lies within the span from the new data point to the intersection point, which is not included in the argument list.

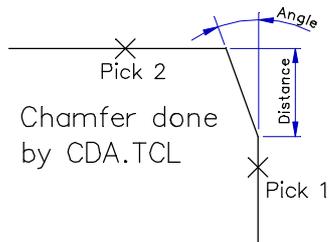
```

*/
static updent(ent,pnt)
ENTITY ent;           // Entity (line) to be updated by a point
POINT pnt;           // Point to update to line entity
{
    ENTDATA seg;

    seg = ent_read(ent);           // Read it

    if ( !in_span(ent,pnt) ) {
        if ( nearpl(seg.line.ps,seg.line.pe,pnt) )
            seg.line.ps = pnt;
        else
            seg.line.pe = pnt;
    }
    else {
        if ( in_span(L(ent.pick,seg.line.ps),pnt) )
            seg.line.ps = pnt;
        else
            seg.line.pe = pnt;
    }
    ent_write(ent,seg);           // Update it
}

```



CLOCK.TCL

The source program listed below is an example for showing a real time clock. The programming technique applied here is quite straightforward. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```

/*****
    Program:      CLOCK.TCL
    Author:       Kang, Hsin-Min
    Purpose:      Demonstration of TCL programming
    Effect:       Showing the ticktack of a clock
*****/

#define hangle    (-chour*30+90-cmin/2)
#define mangle    (-cmin*6+90)

POINT    center;    // Center of the clock
double   ssize;     // Second needle size
double   msize;     // Minute needle size
double   hsize;     // Hour needle size
int      csec;      // Current time in second
int      cmin;      // Current time in minute
int      chour;     // Current time in hour
int      ctime[4];  // Array to read time

COMMAND clock()
{
    int    i;
    int    wnd[]={5,10,2,19,0x374f};
    double size;    // basic size of clock
    double esize;   // 1-minute-stone size
    double dsize;   // 5-minute-stone size
    double angle;   // Temporary variable

    @cmdecho = 0;
    command("Record OFF");

    center = (@vwndmin+@vwndmax)/2;
    size = @vwndmax.y - @vwndmin.y;
    size /= 3;
    ssize = size/5;
    msize = size;
    hsize = 3*size/5;
    dsize = size/40;
    esize = dsize/1.5;
    c1 = center,size*1.1;
    c2 = center,size*1.05;
    c4 = center,ssize;

    // Prepare the clock surface

    plotent(c(center,size*1.2)); // Rim of clock
    plotent(c(center,dsize));   // Hub of clock

    setplot(0x07,0,0,0);
    for(i=0;i<60;i++) {
        if ( i % 5 ) // 5-minute stone
            plotent(c(p(c1,(i*6)A),esize));
        else {
            setplot(0x0e,0,0,0); // 1-minute stone
        }
    }
}

```

```

        plotent(c(p(c1,(i*6)A),dsize));
        setplot(0x07,0,0,0);
    }
}

time(ctime);
chour = ctime[0];
cmin = ctime[1];
setplot(0x8e,0,0,0);          // XOR mode, Yellow
l3 = center,p(c(center,hsize),(hangle)A);
plotent(l3);                  // Plot hour needle

setplot(0x8e,0,0,0);          // XOR mode, Yellow
l2 = center,p(c(center,msize),(mangle)A);
plotent(l2);                  // Plot minute needle

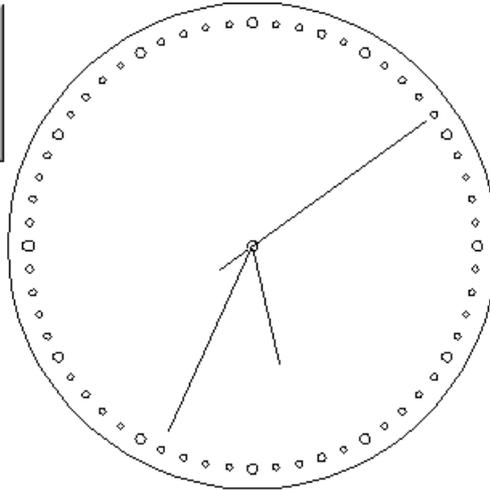
setplot(0x8f,0,0,0);          // XOR mode, white
csec = ctime[2];
angle = 90-csec*6;
l1 = p(c4,(angle-180)A),p(c2,(angle)A);
plotent(l1);                  // Plot second needle
wndopen(wnd,"Current Time",0x321e);
userbrk(0);
for(;;) {
    if ( userbrk(-1) )          // Has user break?
        break;
    s1=time(ctime);             // Get current time
    if ( csec == ctime[2] )     // Change in second?
        continue;
    plotent(l1);                // Erase last one
    csec = ctime[2];
    angle = 90-csec*6;
    l1 = p(c4,(angle-180)A),p(c2,(angle)A);
    plotent(l1);                // Plot new second needle
    if ( csec == 0 )            // Across 60?
        updmin();
    wnd_loct(2,1);              // Show time in window
    wnd_puts(left$(s1,8),0x304f);
//    printf("\n%s",left$(s1,8));
}
windclose();
command("Record ON");
@cmdecho = 1;
}

void updmin()
{
    setplot(0x8e,0,0,0);
    plotent(l2);                // Erase last
    cmin = ctime[1];
    l2 = center,p(c(center,msize),(mangle)A);
    plotent(l2);                // Draw new
    setplot(0x8f,0,0,0);
    if ( (cmin%5)==0 )          // Update hour needle at
        updhour();              // every 5 minutes
}

void updhour()
{
    setplot(0x8e,0,0,0);
    plotent(l3);                // Erase last

```

```
    chour = ctime[0];  
    l3 = center,p(c(center,hsize),(hangle)A);  
    plotent(l3); // Draw new  
    setplot(0x8f,0,0,0);  
}
```



FILES.TCL

```
/*
Command: FILES -- A General Purpose File Manager
Programmed by Kang, Hsin-Min.
First edition in 1995 for TCAM/TurboCAD (TwinCAD's DOS version)
Revised on Sep 13, 1997 for TwinCAD as a standard command:

    1. Add [Help] button for help invocation in Windows.
    2. Change programming style for multiple language support.

Revised on Apr 22, 1998 Include Japanese Text in SJIS codes.
Revised on Aug 05, 1998 Modify Japanese Messages.
Copyright (c) 1995, 1997, 1998 TCAM Development House, Taipei
*/

#define strlen(s1)      len(s1)
#define ESC             -1
#define QUIT           0
#define LoopForever    while(1)

int  IsWin;
int  bList;
int  bCopy;
int  bRename;
int  bErase;
int  bMove;
int  bMCopy;
int  bExit;
int  bHelp;

char  szBtn[12][16];    // For all common buttons
char  szText[20][40];  // Our display text buffer, see SetMsgText()
                        // at the end of this file

COMMAND Files()
{
    int  fnc;

    SetMsgText();
    MainDiaWnd();      // Open the main dialogue window
    LoopForever {
        fnc=wnd_gbtn(0x1000);    // Return any keypress
        switch(fnc) {
            case QUIT:
            case ESC:
            case bExit:
                wndclose();
                exit(0);
                break;
            case bList:   DoListFile();    continue;
            case bCopy:   DoCopyFile();    continue;
            case bRename: DoRename();continue;
            case bErase:  DoDelete();continue;
            case bMove:   DoMoveFile();    continue;
            case bMCopy:  DoCopyDir();     continue;
            case bHelp:   show_help("", "FILES"); continue;
        }
        if ( IsWin ) continue;
    }
}
```

```

        switch(fnc&0xff) { // Check Keyboard codes for DOS version
            case 'L': case 'l': DoListFile(); continue;
            case 'D': case 'd': DoCopyFile(); continue;
            case 'R': case 'r': DoRename(); continue;
            case 'E': case 'e': DoDelete(); continue;
            case 'M': case 'm': DoMoveFile(); continue;
            case 'C': case 'c': DoCopyDir(); continue;
            case 'H': case 'h': Show_help("", "FILES"); continue;
            case 'X': case 'x':
                wndclose();
                exit(0);
        }
    }
}

DoListFile()
{
    getfile("*. *", "", 0, szText[1]/*"Browse Directory Files"*/);
}

DoCopyFile()
{
    char SrcFile[64]; // Source file
    char DstFile[64]; // Destination file
    long nbytes; // Number of Bytes Copied

    SrcFile = getfile("*. *", "", 0, szText[2]);
    /*"Select Single Source File to Copy"*/

    if ( !strlen(SrcFile) ) return 0;

    DstFile = getfile(SrcFile, "", 1, szText[3]);
    /*"Enter the Destination Filename"*/

    if ( !strlen(DstFile) ) return 0;

    CopyOneFile(SrcFile, DstFile);
    beep();
}

DoRename()
{
    char SrcFile[64]; // Source file
    char DstFile[64]; // Destination file
    int i;

    SrcFile = getfile("*. *", "", 0, szText[4]);
    /*"Select File to Rename"*/
    if ( !strlen(SrcFile) ) return 0;

    DstFile = getfile(SrcFile, "", 1, szText[5]);
    /*"Enter the New Filename"*/
    if ( !strlen(DstFile) ) return 0;

    printf("\nRename %s to %s", SrcFile, DstFile);
    if ( takepath(SrcFile) != takepath(DstFile) )
        i = MoveOneFile(Srcfile, DstFile);
    else
        i = rename(SrcFile, DstFile);
    switch(i) {
        case 2: printf("\n## File not found!"); break;
    }
}

```

```

        case 3:    printf("\n## Path not found!"); break;
        case 5:    printf("\n## Access Denied!"); break;
        case 0x11:printf("\n## Not the same device!"); break;
    }
    beep();
}

DoDelete()
{
    char ListFile[64];    // File Name List File
    char SrcFile[65];    // File to erase
    int hdl;
    int state=0;

    ListFile = getfile("*.*", "*", 0x8000, szText[6]);
    /*Select Files to Erase*/
    if ( !strlen(ListFile) ) return 0;
    if ( index(ListFile, ".FNL") ) { // In case old version!
        hdl = fopen(ListFile, "r");
        while(fgets(SrcFile, 64, hdl)>0) {
            if ( state != 2 )
                state = ConfirmFile(szText[11]+SrcFile+" ?");
            if ( state ) {
                printf("\nErase File: %s", SrcFile);
                if ( remove(SrcFile) == 5 )
                    printf(" ## Access Denied!");
            }
        }
    }
    remove(ListFile);
    beep();
}

DoMoveFile()
{
    char ListFile[64];    // File Name List File
    char SrcFile[65];    // File to Move
    char DstFile[65];    // Where to Move
    int hdl;
    int state=0;

    if ( OldVersion() ) return 0;

    ListFile = getfile("*.*", "*", 0x8000, szText[7]);
    /*Select Files to Move */
    if ( !strlen(ListFile) ) return 0;
    DstFile = upper(getpath(takepath(ListFile), 0, szText[8]));
    /*Select Directory to Move Files to*/
    if ( strlen(DstFile) ) {
        hdl = fopen(ListFile, "r");
        while(fgets(SrcFile, 64, hdl)>0) {
            if ( takepath(SrcFile)==DstFile )
                break;
            if ( state != 2 )
                state = ConfirmFile(szText[12]+SrcFile+" ?");
            if ( state ) {
                printf("\nMove File: %s", SrcFile);
                if ( MoveOneFile(SrcFile,
                    fullname(takename(SrcFile), DstFile)) == 5 )
                    printf(" ## Access Denied!");
            }
        }
    }
}

```

```

    }
  }
  remove(ListFile);
  beep();
}

DoCopyDir()
{
  char ListFile[64];      // File Name List File
  char SrcFile[65];      // File to Move
  char DstFile[65];      // Where to Move
  int hdl;
  int state=0;
  long nbytes, tnbytes;

  if ( OldVersion() ) return 0;

  ListFile = getfile("*.*", "*", 0x8000, szText[9]);
  /*"Select Files to Copy"*/
  if ( !strlen(ListFile) ) return 0;
  DstFile = upper(getpath(takepath(ListFile), 0, szText[10]));
  /*"Select Directory to Copy Files to"*/
  if ( strlen(DstFile) ) {
    hdl = fopen(ListFile, "r");
    while(fgets(SrcFile, 64, hdl) > 0) {
      if ( takepath(SrcFile) == DstFile )
        break;
      if ( state != 2 )
        state = ConfirmFile(szText[13] + SrcFile + " ?");
      if ( state ) {
        if ( CopyOneFile(SrcFile,
                        fullname(takename(SrcFile), DstFile)) )
          break;
      }
    }
  }
  remove(ListFile);
  beep();
}

ConfirmFile(char Message[])
{
  int wnd[] = {13, 17, 5, 75, 0x371e};
  int i;

  i = strlen(Message) + 8;
  if ( i < 36 ) i = 36;
  if ( i > 80 )
    i = 80;
  i /= 2;
  wnd[2] = 40 - i;
  wnd[3] = 40 + i - 1;
  beep();
  wndopen(wnd);
  wnd_loct(1, 2);
  wnd_puts(Message, 0x61e);

  WndButton(3, i - 16, 0x108, szBtn[9] /*" NO "*/); // 0
  WndButton(3, i - 4, 0x108, szBtn[8] /*" Yes "*/); // 1
  WndButton(3, i + 8, 0x108, szBtn[10] /*" All "*/); // 2
}

```

```
LoopForever {
    i = wnd_gbtn(0x1000);
    switch(i&0xff) {
        case 0: case 1: case 2: break;
        case 'N': case 'n': i = 0; break;
        case 'Y': case 'y': i = 1; break;
        case 'A': case 'a': i = 2; break;
        default: continue;
    }
    wndclose();
    return i;
}

OldVersion()
{
    if ( !identify("getpath") ) {
        printf("\n## Your Version is too old to have the 'getpath()'
function support!");
        beep();
        return 1;
    }
    return 0;
}

MoveOneFile(char SrcFile[],char DstFile[])
{
    if ( copyfile(SrcFile,DstFile) < 0 )
        return 5;
    return remove(SrcFile);
}

CopyOneFile(char SrcFile[],char DstFile[])
{
    long nbytes;
    if ( identify("copyfile") ) {
        printf("\nCopy %s to %s",SrcFile,DstFile);
        nbytes = copyfile(SrcFile,DstFile);
        if ( nbytes > 0 )
            printf("\nOperation Succeeded, %ld bytes copied.",nbytes);
        else if ( nbytes == -1L ) {
            printf("\n## Source File Error!"); return 1;
        }
        else if ( nbytes == -2L ) {
            printf("\n## Destination File Error!"); return 1;
        }
    }
    else
        command("DOS COPY \@SrcFile \@DstFile^MEXIT^M");
    return 0;
}

MainDiaWnd()
{
    int wnd[]={10,18,13,63,0x372f};
    int rBase, cBase;

    SetMsgText();

    if ( IsWin )
        wnd[4] = 0x3271;
}
```

```

        wndopen(wnd,szText[0]/*"File Utilities"*/,0x321e,0x0000);

        rBase = 1;
        cBase = 2;
        bList = WndButton(rBase, cBase, 0x10e,szBtn[0]/*" List files
**/);
        bCopy = WndButton(rBase, cBase+16,0x10e,szBtn[1]/*" Duplicate
**/);
        bRename = WndButton(rBase, cBase+32,0x10e,szBtn[2]/*" Rename
file"*/);
        bErase = WndButton(rBase+2,cBase, 0x10e,szBtn[3]/*" Erase
files"*/);
        bMove = WndButton(rBase+2,cBase+16,0x10e,szBtn[4]/*" Move files
**/);
        bMCopy = WndButton(rBase+2,cBase+32,0x10e,szBtn[5]/*" Copy
files "*/);

        if ( IsWin ) {
            bExit = WndButton(rBase+4,cBase+10,0x108,szBtn[6]/*" Exit
**/);
            bHelp = WndButton(rBase+4,cBase+26,0x108,szBtn[7]/*" Help
**/);
        }
        else { // No help in DOS version!
            bExit = WndButton(rBase+4,cBase+20,0x108,szBtn[6]/*" Exit
**/);
        }
    }
    //
    // Some useful routine supports
    //
    int nSupport=identify("wnd_resize");
    //
    int WndButton(int r,int c, int bsize, char text[])
    {
        int i;
        if ( nSupport ) // Having direct support ?
            return wnd_button(r,c,bsize,0x1277,text);
        wnd_loct(r,c);
        i = wnd_field(0,IsWin?0x1277:0x370,bsize);
        wnd_puts(text,0x270);
        return i;
    }

char szAsciiText[][40]={ // Default English Display Text
/* 0 */ "File Utilities",
/* 1 */ "Browse Directory Files",
/* 2 */ "Select Single Source File to Copy",
/* 3 */ "Enter the Destination Filename",
/* 4 */ "Select File to Rename",
/* 5 */ "Enter the New Filename",
/* 6 */ "Select Files to Erase",
/* 7 */ "Select Files to Move",
/* 8 */ "Select Directory to Move Files to",
/* 9 */ "Select Files to Copy",
/* 10 */ "Select Directory to Copy Files to",
/* 11 */ "Erase ",
/* 12 */ "Move ",
/* 13 */ "Copy ",

```

```
};

char szBig5Text[][40]={          // Default Chinese Big-5 Display Text
/* 0 */ "檔案管理操作",
/* 1 */ "檔案目錄導覽",
/* 2 */ "請選擇要複製的檔案",
/* 3 */ "請輸入要複製成的檔案",
/* 4 */ "請選擇要改名的檔案",
/* 5 */ "請輸入新的檔名路徑",
/* 6 */ "請選取要刪除的檔案",
/* 7 */ "請選取要搬移的檔案",
/* 8 */ "請指定搬移的目的地路徑",
/* 9 */ "請選取要拷貝的檔案",
/* 10 */ "請指定拷貝的目的地路徑",
/* 11 */ "您確定要刪除 ",
/* 12 */ "您確定要搬移 ",
/* 13 */ "您確定要拷貝 ",
};

char szGB2312Text[][40]={       // Default Chinese GB2312 Display Text
/* 0 */ "紫偶奪燴紱釵",
/* 1 */ "紫偶醴翹絳擬",
/* 2 */ " 恁 寔 猗 葩 柰 腔 紫 偶",
/* 3 */ " 怀 猗 葩 柰 儉 腔 紫 偶",
/* 4 */ " 恁 寔 猗 蝻 靡 腔 紫 偶",
/* 5 */ " 怀 猗 腔 紫 靡 繚 嚟",
/* 6 */ " 恁 猗 刼 壺 腔 紫 偶",
/* 7 */ " 恁 猗 唸 疔 腔 紫 偶",
/* 8 */ " 硤 隅 唸 疔 腔 醴 華 繚 嚟",
/* 9 */ " 恁 猗 蕭 探 腔 紫 偶",
/* 10 */ " 硤 隅 蕭 探 腔 醴 華 繚 嚟",
/* 11 */ "蠟 隅猗刼壺 ",
/* 12 */ "蠟 隅猗唸疔 ",
/* 13 */ "蠟 隅猗蕭探 ",
};
//
// The following Japanese messages are translated by _____.
//
char szSJISText[][40]={         // Default Japanese SJIS Display Text
/* 0 */ "恸紱*",
/* 1 */ "蹬侖蜎*",
/* 2 */ "瞄箭 恸紱",
/* 3 */ " ** 恸紱",
/* 4 */ " 恸紱",
/* 5 */ " 恸紱",
/* 6 */ " ** 恸紱",
/* 7 */ " 恸紱",
/* 8 */ "恸紱 蹬侖蜎",
/* 9 */ "瞄箭 恸紱",
/* 10 */ "恸紱瞄箭 蹬侖蜎",
/* 11 */ " **",
/* 12 */ "",
/* 13 */ "瞄箭",
};
//
```

```
// The following Korean messages are translated by _____.
//
char szKoreanText[][40]={ // Default Korean Display Text
/* 0 */ "File Utilities",
/* 1 */ "Browse Directory Files",
/* 2 */ "Select Single Source File to Copy",
/* 3 */ "Enter the Destination Filename",
/* 4 */ "Select File to Rename",
/* 5 */ "Enter the New Filename",
/* 6 */ "Select Files to Erase",
/* 7 */ "Select Files to Move",
/* 8 */ "Select Directory to Move Files to",
/* 9 */ "Select Files to Copy",
/* 10 */ "Select Directory to Copy Files to",
/* 11 */ "Erase ",
/* 12 */ "Move ",
/* 13 */ "Copy ",
};

char szAsciiBtn[][16]={ // Default English Display Text
/* 0 */ "&List files",
/* 1 */ "&Duplicate",
/* 2 */ "&Rename file",
/* 3 */ "&Erase files",
/* 4 */ "&Move files",
/* 5 */ "&Copy files",
/* 6 */ "E&xit",
/* 7 */ "&Help",
/* 8 */ "&Yes",
/* 9 */ "&No",
/* 10 */ "&All",
};

char szBig5Btn[][16]={ // Default English Display Text
/* 0 */ "檔案目錄[&L]",
/* 1 */ "單一拷貝[&D]",
/* 2 */ "更改檔名[&R]",
/* 3 */ "多檔刪除[&E]",
/* 4 */ "多檔搬移[&M]",
/* 5 */ "多檔拷貝[&C]",
/* 6 */ "結束[&X]",
/* 7 */ "說明[&H]",
/* 8 */ "確定[&Y]",
/* 9 */ "不要[&N]",
/* 10 */ "都要[&A]",
};

char szGB2312Btn[][16]={ // Default English Display Text
/* 0 */ "紫偶醴翹[&L]",
/* 1 */ "等玲蕭探[&D]",
/* 2 */ "載蜎紫靡[&R]",
/* 3 */ "嗣紫刳壺[&E]",
/* 4 */ "嗣紫唸庠[&M]",
/* 5 */ "嗣紫蕭探[&C]",
/* 6 */ "賦吁[&X]",
/* 7 */ "依隴[&H]",
/* 8 */ "隅[&Y]",
/* 9 */ "祥猊[&N]",
};
```

```
/* 10 */ "飲獺[&A]",
};

char szSJISBtn[][16]={ // Default English Display Text
/* 0 */ "婁壞&L",
/* 1 */ "瞄箭[&D]",
/* 2 */ "恹絨 [ &R]",
/* 3 */ " * &E",
/* 4 */ " [ &M]",
/* 5 */ " ** 瞄箭[&C]",
/* 6 */ " [ &X]",
/* 7 */ "陆忝[&H]",
/* 8 */ "&Yes",
/* 9 */ "&No",
/* 10 */ " * &A",
};

char szKoreanBtn[][16]={ // Default English Display Text
/* 0 */ "&List files",
/* 1 */ "&Duplicate",
/* 2 */ "&Rename file",
/* 3 */ "&Erase files",
/* 4 */ "&Move files",
/* 5 */ "&Copy files",
/* 6 */ "E&xit",
/* 7 */ "&Help",
/* 8 */ "&Yes",
/* 9 */ "&No",
/* 10 */ "&All",
};

char szDosAsciiBtn[][16]={ // Default English Display Text
/* 0 */ " List files ",
/* 1 */ " Duplicate ",
/* 2 */ " Rename file",
/* 3 */ " Erase files",
/* 4 */ " Move files ",
/* 5 */ " Copy files ",
/* 6 */ " Exit ",
/* 7 */ " Help ",
/* 8 */ " Yes ",
/* 9 */ " No ",
/* 10 */ " All ",
};

char szDosBig5Btn[][16]={ // Default English Display Text
/* 0 */ " L:檔案目錄 ",
/* 1 */ " D:單一拷貝 ",
/* 2 */ " R:更改檔名 ",
/* 3 */ " E:多檔刪除 ",
/* 4 */ " M:多檔搬移 ",
/* 5 */ " C:多檔拷貝 ",
/* 6 */ " X:結束 ",
/* 7 */ " H:說明 ",
/* 8 */ " Y:確定 ",
/* 9 */ " N:不要 ",
/* 10 */ " A:都要 ",
};
```

```
char  szDosGB2312Btn[][16]={          // Default English Display Text
/* 0 */ " L:紫偶醴翹 ",
/* 1 */ " D:等玲蕭探 ",
/* 2 */ " R:載蜎紫靡 ",
/* 3 */ " E:嗣紫刳壺 ",
/* 4 */ " M:嗣紫唵炸 ",
/* 5 */ " C:嗣紫蕭探 ",
/* 6 */ " X:賦吁 ",
/* 7 */ " H:飲隴 ",
/* 8 */ " Y: 隅 ",
/* 9 */ " N:祥猊 ",
/* 10 */ " A:飲猊 ",
};
```

```
char  szDosSJISBtn[][16]={          // Default English Display Text
/* 0 */ "L: 恸紱",
/* 1 */ "D: 恸紱 瞄箭",
/* 2 */ "R: 恸紱      ",
/* 3 */ "E: 恸紱    **",
/* 4 */ "M: 恸紱      ",
/* 5 */ "C: 恸紱 瞄箭",
/* 6 */ "E:      ",
/* 7 */ "H: 陆杺",
/* 8 */ "Y:Yes",
/* 9 */ "N:No",
/* 10 */ "A:  **",
};
```

```
char  szDosKoreanBtn[][16]={       // Default English Display Text
/* 0 */ " List files ",
/* 1 */ " Duplicate ",
/* 2 */ " Rename file",
/* 3 */ " Erase files",
/* 4 */ " Move files ",
/* 5 */ " Copy files ",
/* 6 */ " Exit ",
/* 7 */ " Help ",
/* 8 */ " Yes ",
/* 9 */ " No ",
/* 10 */ " All ",
};
```

```
#define BIG5          0x404
#define GB2312       0x804
#define SJIS         0x411
#define KOREAN       0x412
#define ASCII        0x409
```

```
#define TGBIG5       1
#define TGGB2312    2
#define TGSJIS       5
#define TGKOREAN     3
```

```
int  SetMsgText()
{
    char  szBuf[80];
    int   switched;
```

```

IsWin = 1;
switch(1) { // Test if we are in TwinCAD ...
  case !identify("os_type"):
  case (os_type()<0): // Must be in DOS version!
    IsWin = 0;
    break;
}

setdata(szText,szAsciiText);
if ( IsWin ) {
  szBuf = get_profile("General Setup","AlternateLanguage");
  switched = 0;
  if ( szBuf!=" " )
    switched = eval(szBuf);
  setdata(szBtn,szAsciiBtn);
  if ( switched ) {
    switch(os_language()) {
      case BIG5: setdata(szText,szBig5Text); break;
      case GB2312: setdata(szText,szGB2312Text); break;
      case SJIS: setdata(szText,szSJISText); break;
      case KOREAN: setdata(szText,szKoreanText); break;
    }
    switch(os_language()) {
      case BIG5: setdata(szBtn,szBig5Btn); break;
      case GB2312: setdata(szBtn,szGB2312Btn); break;
      case SJIS: setdata(szBtn,szSJISBtn); break;
      case KOREAN: setdata(szBtn,szKoreanBtn); break;
    }
  }
}
else { // Switched or not, depends on TGF type in DOS
version
  setdata(szBtn,szDosAsciiBtn);
  switch(@tgftype) {
    case TGBIG5: setdata(szText,szBig5Text); break;
    case TGGB2312: setdata(szText,szGB2312Text); break;
    case TGSJIS: setdata(szText,szSJISText); break;
    case TGKOREAN: setdata(szText,szKoreanText); break;
  }
  switch(@tgftype) {
    case TGBIG5: setdata(szBtn,szDosBig5Btn); break;
    case TGGB2312: setdata(szBtn,szDosGB2312Btn); break;
    case TGSJIS: setdata(szBtn,szDosSJISBtn); break;
    case TGKOREAN: setdata(szBtn,szDosKoreanBtn); break;
  }
}
return 0;
}

```



HANOI.TCL

The source program listed below is an example for solving the Hanoi Tower problem. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed. The programming technique applied here can be found in most books on programming. Nevertheless, the concept used for solving this problem is outlined as below:

To move n disks from ONE pack to ANOTHER pack, we must firstly,

- (1) move the upper $n-1$ disks from ONE pack to THE OTHER pack, then,
- (2) move the n^{th} disks from ONE pack to ANOTHER pack, and finally,
- (3) move the rest of $n-1$ disks from THE OTHER pack to ANOTHER pack.

```

/*****
    Program:      HANOI.TCL
    Author:       Kang, Hsin-Min
    Purpose:      Demonstration of TCL programming
    Effect:       Solving the Hanoi Tower problem with visual effect.
*****/

int  pack[3];      /* Store the number of disks in each pack. */
int  disk[3][10]; /* Pack stack to store the disks. */
int  ndisk;        /* Number of disk to move */
int  nmove;        /* Number of move */
int  delay=2;     /* Short delay for visual retention */
int  packtop;
int  color[]={0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x07,0x06,0x02,0x0f};

#define BVSIZE      4      /* Vertical size of block */
#define PACKPOS1    18
#define PACKPOS2    40
#define PACKPOS3    62
#define GNDLEVEL    0      /* Ground level */
#define LMARGIN     0
#define RMARGIN     (LMARGIN+80)

COMMAND Hanoi()
{
    int n, c;

    @cmdecho = 0;
    p1 = LMARGIN,GNDLEVEL-5.;
    p2 = RMARGIN,GNDLEVEL+12*BVSIZE;
    command("zw p1 p2 "); // Set up window

    for (;;) {
        if ((ndisk=getv("\nPlease Enter Number of Disks to Demo
<0..9>:",3))==0 )
            break;
        if ( ndisk > 9 )
            break;
        delay = getv("\nPlease enter delay count (5): ",5);
        command("^N "); // Renew the screen
        n = ndisk-1;
        initblk(n);
        pack[0] = ndisk;
        pack[1] = pack[2] = nmove = 0;
    }
}

```

```

        for (n=0; n<ndisk; n++) {
            disk[0][n] = n;
            disk[1][n] = disk[2][n] = 0;
        }
        movedisk(n,1,3);
    }
    @cmdecho = 1;
}

movedisk(n,pack1,pack2)
int  n;          /* The number of disk in pack to be moved. */
int  pack1;     /* The pack containing the disks to be moved off. */
int  pack2;     /* The pack where the disk to be moved to. */
{
    if ( n == 1 )
        moveone(pack1,pack2);
    else {
        movedisk(n-1,pack1,6-pack1-pack2);
        movedisk(1,pack1,pack2);
        movedisk(n-1,6-pack1-pack2,pack2);
    }
}

static      int    packposh[3]={PACKPOS1, PACKPOS2, PACKPOS3};

moveone(pack1,pack2)
int  pack1;     /* pack number where the disk to be moved from. */
int  pack2;     /* pack number where the disk to be moved to. */
{
    int  h1,h2;      /* absolute horizontal position of pack. */
    int  v1,v2;      /* relative vertical position of disk in pack.
*/
    int  disksize;  /* size number of the disk to be moved. */
    int  n1, n2;

    n1 = pack1-1;
    n2 = pack2-1;
    h1 = packposh[n1];
    h2 = packposh[n2];
    v1 = pack[n1]-1;
    v2 = pack[n2];
    disksize = disk[n1][v1];
    printf("\n%d> Moving one disk (%d) from %c to %c ...",
        ++nmove,disksize,n1+'A',n2+'A');

    liftblk(v1*BVSIZE,disksize,h1);
    moveblk(disksize,h1,h2);
    putblk(v2*BVSIZE,disksize,h2);

    disk[n2][v2] = disksize;
    pack[n1]--;
    pack[n2]++;
}

holdit()
{
    int  i;
    for (i=0; i<delay; i++);
}

initblk(n)

```

```

int  n;    /* number of disk for this demo. */
{
    int  i;
    int  h;

    setplot(0x0e,0,0,0);

    h = (n+1)*BVSIZE+BVSIZE/2;

    packtop = h+BVSIZE/2;

    plotent(L(P(LMARGIN,GNDLEVEL),P(RMARGIN,GNDLEVEL)));
    plotent(L(P(PACKPOS1,GNDLEVEL),P(PACKPOS1,h)));
    plotent(L(P(PACKPOS2,GNDLEVEL),P(PACKPOS2,h)));
    plotent(L(P(PACKPOS3,GNDLEVEL),P(PACKPOS3,h)));

    setplot(0x8f,0,0,0);
    for (i=0; i<=n; i++)
        drawblk(i*BVSIZE,i,PACKPOS1);
}

moveblk(disksize,j1,j2)
/*
    Routine to move disk on screen horizontally.
    Vertical position is fixed.
*/
int  disksize; /* size of the disk to be moved. */
int  j1;      /* absolute horizontal position to start moving disk. */
int  j2;      /* absolute horizontal position to end moving disk. */
{
    int k,n;

    if ( j1 < j2 ) {
        for (k=j1; k<j2; k++) {
            holdit();
            plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
            p1.x++; p2.x++; p3.x++; p4.x++;
            plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
        }
    }
    else {
        for ( k=j1; k>j2; k--) {
            holdit();
            plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
            p1.x--; p2.x--; p3.x--; p4.x--;
            plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
        }
    }
}

liftblk(diskpos,disksize,packpos)
/*
    Routine to lift disk vertically on screen.
*/
int  diskpos; /* vertical position of the disk to be moved. */
int  disksize; /* size of disk to be moved. */
int  packpos; /* absolute horizontal pack positon for the disk. */
{
    int pos,n;

    setblk(diskpos,disksize,packpos);
}

```

```
        for (pos=diskpos; pos<packtop; pos++) {
            holdit();
            plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
            p1.y++; p2.y++; p3.y++; p4.y++;
            plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
        }
}

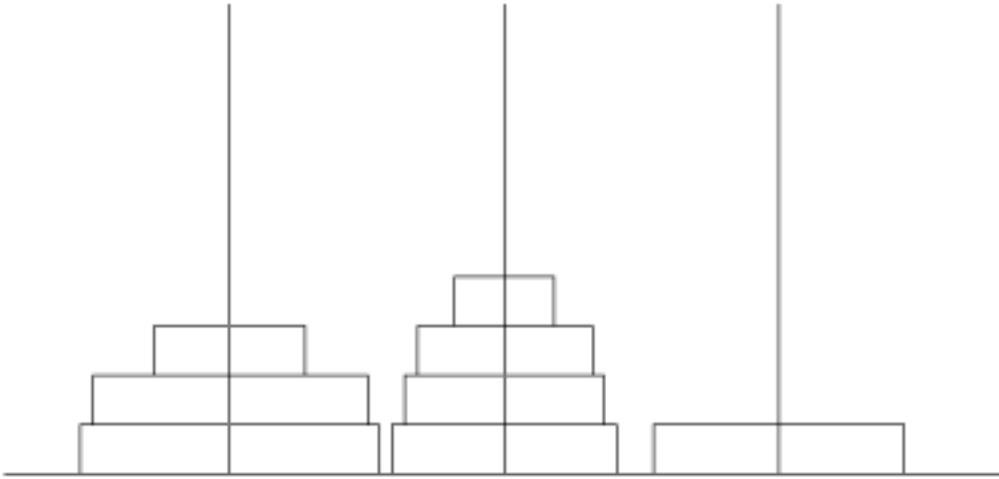
putblk(diskpos,disksize,packpos)
/*
    Routine to put disk down vertically on screen.
*/
int  diskpos;    /* vertical position of the disk to be moved. */
int  disksize;  /* size of disk to be moved. */
int  packpos;   /* absolute horizontal pack positon for the disk. */
{
    int pos,n;

    for (pos=packtop; pos>diskpos; pos--) {
        holdit();
        plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
        p1.y--; p2.y--; p3.y--; p4.y--;
        plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
    }
}

drawblk(diskpos,disksize,packpos)
int  diskpos;    /* vertical position of the disk to be moved. */
int  disksize;  /* size of disk to be moved. */
int  packpos;   /* absolute horizontal pack positon for the disk. */
{
    setblk(diskpos,disksize,packpos);
    plotent(L(p1,p2),L(p2,p3),L(p3,p4),L(p4,p1));
}

setblk(diskpos,disksize,packpos)
int  diskpos;    /* vertical position of the disk to be moved. */
int  disksize;  /* size of disk to be moved. */
int  packpos;   /* absolute horizontal pack positon for the disk. */
{
    int k, m, n;
    int  size;

    setplot(0x80|color[disksize],0,0,0);
    size = ndisk - disksize + 3;
    v1 = diskpos+GNDLEVEL;
    v2 = v1+BVSIZE;
    v3 = packpos-size;
    v4 = packpos+size;
    p1 = v3,v1;
    p2 = v4,v1;
    p3 = v4,v2;
    p4 = v3,v2;
}
}
```



LRDIM.TCL

The source program listed below is an example for creating a Large Radius Dimensioning with bending lines. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```

/*****
    Command:      LRDIM -- Create Large Radius Dimension Command.

    Programmed by Kang, Hsin-Min.
    Release on Aug 26, 1994 -- As a working example in DOS version
    Revised on Feb 10, 1998 -- Add multiple language support,
        include as a standard command.
    Revised on Apr 22, 1998 Include Japanese Text in SJIS codes
    Revised on Aug 05, 1998 Modify Japanese Messages.

    Copyright (c) 1994, 1998 TCAM Development House, Taipei
    All rights Reserved.

    Status:      TwinCAD Standard Command
    Comments:    This program creates leader with text to produce ANSI standard
        of Large radius dimensioning with bending lines.
*****/

#include <tclcad.h>
#include <geofnc.h>
#define ppangle(ps,pe) atan2((pe).y-(ps).y,(pe).x-(ps).x)
#define FARC 8
#define DIMRAD 3
#define DIMTCEN 0x08 /* PT specify center position */
#define DIMTEXT 0x8000 /* Use local text */

#define TGBIG5 1
#define TGGB2312 2
#define TGSJIS 5
#define TGKOREAN 3
char szText[5][70]; /* Our display text buffer
char szAsciiText[][60]={ /* Default English Display Text
/* 0 */ "\nLargeRadiusDIM -- Select arc segment to dimension: ",
/* 1 */ "\nLargeRadiusDIM -- Indicate the starting point: ",
/* 2 */ "\n** Invalid point. You should use RDIM...",
};
char szBig5Text[][60]={ /* Default Chinese Big-5 Display Text
/* 0 */ "\n 大半徑標註 -- 請選取要進行標註的圓弧: ",
/* 1 */ "\n 大半徑標註 -- 請指定標註起點: ",
/* 2 */ "\n** 起點位置不適切. 請轉用基本半徑標註...",
};
char szGB2312Text[][60]={ /* Default Chinese GB2312 Display Text
/* 0 */ "\n 湮圍噤梓韶 -- 恁 猓輛賃梓韶腔埴說: ",
/* 1 */ "\n 湮圍噤梓韶 -- 硤隅梓韶 莢: ",
/* 2 */ "\n** 莢龔离祥莖 . 蚩斲價掛圍噤梓韶...",
};
//
// The following Japanese messages are translated by _____.
//
char szSJISText[][60]={ /* Default Japanese SJIS Display Text
/* 0 */ "\n -- : ",
/* 1 */ "\n : ",
/* 2 */ "\n** ** (RDIM) ",

```

```

};
//
// The following Korean messages are translated by _____.
//
char szKoreanText[][60]={ // Default Korean Display Text
/* 0 */ "\nLargeRadiusDIM -- Select arc segment to dimension: ",
/* 1 */ "\nLargeRadiusDIM -- Indicate the starting point: ",
/* 2 */ "\n** Invalid point. You should use RDIM...",
};

COMMAND LRdim()
{
    ENTDATA data;
    int cmdecho;

    switch(@tgftype) {
        case TGBIG5: setdata(szText,szBig5Text); break;
        case TGGB2312: setdata(szText,szGB2312Text); break;
        case TGSJIS: setdata(szText,szSJISText); break;
        case TGKOREAN: setdata(szText,szKoreanText); break;
        default: setdata(szText,szAsciiText); break;
    }

    e1 = gete(szText[0],FARC);
    p1 = getp(szText[1]);
    data = ent_read(e1);
    if ( V(p1,e1.pick) >= data.arc.rad) {
        printf(szText[2]);
        command("RDIM e1");
        exit();
    }
    if ( V(p1,data.arc.pc) >= data.arc.rad) {
        printf(szText[2]);
        command("RDIM e1");
        exit();
    }
    cmdecho = @cmdecho; // save @cmdecho
    @cmdecho = 0; // Turn off cmdecho
    p4 = e1.pick; // arrow position
    p2 = p1,L(data.arc.pc,e1.pick);
    p3 = (2*p2+p4)/3;
    p2 = p3,L((p3+p4)/2,p1);
    command("Leader p4 p3 p2 p1 ");
    @cmdecho = cmdecho;

    // Prepare default dimension text

    // s1 = @dimrbstr+prints("%0f",data.arc.rad)+@dimrestr;

    s1 = dim_text(DIMRAD,data.arc.rad);

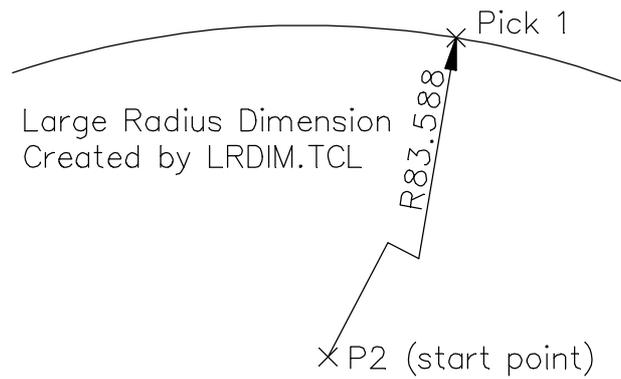
    e2 = ent_last(); // Obtain the leader
    data = ent_read(e2);

    data.dim.flag |= DIMTEXT|DIMTCEN; // Indicating local text
    data.dim.str = s1;
    data.dim.pe = data.dim.ps; // Switch the node points
    data.dim.ps = data.dim.pt;

    data.dim.unit = ppangle(data.dim.pds,data.dim.pde);
    if ( (data.dim.unit>90) && (data.dim.unit<=270) )

```

```
data.dim.unit -= 180;  
  
data.dim.pt =  
(data.dim.pds+data.dim.pde)/2+p(@dimgap,(data.dim.unit+90)A);  
data.dim.unit = dtr(data.dim.unit); // Text angle  
  
ent_write(e2,data);  
exit();  
}
```



LTEXT.TCL

The source program listed below is an example for showing the generation of an extended command which allows the users to create texts with leader. The programming technique applied here is quite straightforward. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```

/*****
Program:      LTEXT.TCL
Author:       Kang, Hsin-Min
Purpose:      Demonstration of TCL programming
Effect:       Build LTEXT Command to Create Texts with leader
*****/

#include <tclcad.h>

COMMAND LText()
{
    int      osmode;      // Used to save @osmode
    int      cmdecho;     // Used to save @cmdecho
    double   width;      // Width of string
    POINT    pst;        // Start point of leader
    POINT    pnd;        // End point of leader
    ENTITY   ent;        // Leader Entity
    ENTDATA  edata;      // Leader data
    char     str[80];     // Test string after leader
    POINT    pstart;     // Start point of text

    osmode = @osmode;    // save @osmode
    cmdecho = @cmdecho;  // save @cmdecho
    userbrk(0);         // Turn off User Break

    for(;;) {           // Loop until ^C or ...
        @cmdecho = 1;
        @osmode = OSGETP|OSNEAR;
        command("Leader"); // Use leader command
        if ( userbrk(-1) ) // Has user break ?
            break;
        ent = ent_last(); // Get just created leader
        if ( ent_type(ent) != T_DIM )
            break;        // Not DIM creation ?
        edata = ent_read(ent); // Read the data
        if ( (edata.dim.type&0x0f) != DIMLEAD )
            break;        // Is it our creation ?
        switch(edata.dim.atype&0x0f) {
            case 5:        // Check number of points ?
                pnd = edata.dim.pe;  pst = edata.dim.ps;
                break;
            case 4:
                pnd = edata.dim.ps;  pst = edata.dim.pt;
                break;
            case 3:
                pnd = edata.dim.pt;  pst = edata.dim.pde;
                break;
            case 2:
                pnd = edata.dim.pde; pst = edata.dim.pds;
                break;
            default:       // Not possible error
                printf("\nImpossible error!");
                beep();
        }
    }
}

```

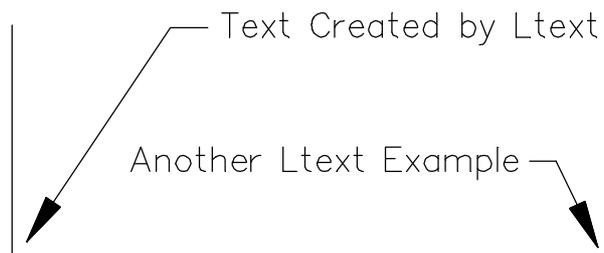
```
        continue;
    }

    str = gets("\nText:",72); // Get inserted text

    pstart.y = pnd.y-@txtheight/2;
    if ( pnd.x >= pst.x ) // Toward Right
        pstart.x = pnd.x+@dimgap;
    else { // Toward left
        width = text_span(str); // Get display width of text
        pstart.x = pnd.x-width-@dimgap;
    }
    @cmdecho = 0; // Turn off command echo
    @osmode = 0;

//    command("TEXT \@pstart 0 @!\@str","\n");

    p1 = pstart; // Use register to avoid nested
    s1 = str; // Evaluation, Faster
    command("TEXT p1 0 @!s1","\n");
}
@cmdecho = cmdecho; // Restore system variables
@osmode = osmode;
}
```



NITEM.TCL

The source program listed below is an example for showing the generation of an extended command which allows the users to create circled item numbers with leader. The programming technique applied here is quite straightforward. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```
/* *****  
Program:      NITEM.TCL  
Author:      Kang, Hsin-Min
```

```

switch(edata.dim.atype&0x0f) {
  case 5:          // Check number of points ?
    pnd = edata.dim.pe;  pst = edata.dim.ps;
    break;
  case 4:
    pnd = edata.dim.ps;  pst = edata.dim.pt;
    break;
  case 3:
    pnd = edata.dim.pt;  pst = edata.dim.pde;
    break;
  case 2:
    pnd = edata.dim.pde; pst = edata.dim.pds;
    break;
  default:        // Not possible error
    printf("\nImpossible error!");
    beep();
    continue;
}

str = prints("%d",counter); // Get string

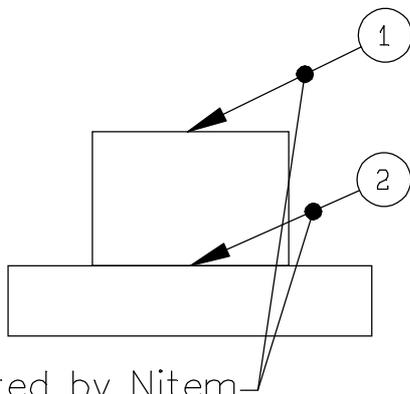
cext = gextent(str); // get text extent
radius = ent_len(cext)/1.9;
if ( radius < @txtheight )
    radius = @txtheight;    // Determine Radius

center = P(L(pnd,pst),-radius);
newent(C(center,radius)); // Create circle

@cmdecho = 0; // Turn off command echo
@osmode = 0;

p1 = center; // Use register to avoid nested
s1 = str;    // Evaluation, Faster
command("TEXT J HC p1 0 @!s1","\n");
counter++;  // Increment count
@NITEMCNT=counter;
}
@cmdecho = cmdecho; // Restore system variables
@osmode = osmode;
}

```



PISTON.TCL

The source program listed below is an example for showing the implementation of a Piston Motion Simulation. The program body as well as the PISTON.WRK drawing are included with TwinCAD CD-ROM and can be found in TwinCAD\TCL and SAMPLE folders when installed. The programming technique applied will be explained briefly as follows.

A drawing named PISTON.WRK was prepared with pre-defined shape, size and position for the flying wheel, crank linkage, piston and the piston cylinder. Each component is placed at different layer. The flywheel is connected to the piston through a crank linkage. While the flywheel rotates about its center counterclockwise, the piston will be sliding back and forth in the cylinder.

In order to calculate the relative motion within the mechanism, you need to specify a reference point for each of the components. This can be done by setting User Variables with **uservar()** function calls. Three user variables are used to store the required information and are saved with the drawing.

Before running the simulation, the program will check to determine whether PISTON.WRK drawing is loaded, and will do searching and load it from disk if it is not. The procedure is fairly straightforward that only specific variables need to be checked.

The program will then generate the drawing images for each of the components, and get into a loop which continuously draws and erases the images for each step of the flywheel movement, until the user types to break. At each step, the program will calculate and setup the required transformation for the drawing images before drawing it out.

```

/*****
    Title:      Piston Motion Simulation
    Purpose:    Showing the Power of TCL Programming
    Author:     Kang, Hsin-min
    Copyright ©1993 By TCAM Development House, Taipei, R.O.C.
    Revised:    March 1, 1997 for TwinCAD
*****/

#define division 5      /* Step angle of the flying wheel */
COMMAND Piston()
{
    double r;           // Distance from crank joint to wheel center
    double b;           // Length of the crank (center distance)
    double c;           // Used in Calculation
    SLIST wheel;        // The part drawing of wheel
    SLIST crank;        // The part drawing of crank
    SLIST piston;       // The part drawing of piston
    POINT wcen;         // Center of wheel
    POINT wrot;         // Wheel Rotation (Cos.Sin)
    POINT crot;         // Crank Rotation (Cos.Sin)

    if ( version()<=101 ) { // If library too old ...
        printf("\nNeed TCL V1.02+ Runtime Library!");
        exit();
    }
    if ( !uservar("WHEELCEN") ) { // Check our foot print ...
        s1=fsearch("PISTON.WRK");
        if ( s1 == "" ) {
            printf("\nThis program requires the PISTON.WRK.");
            exit();
        }
        command("NEW L @!s1");
        if ( !uservar("WHEELCEN") ) {

```

```

        printf("\nThis PISTON.WRK Is Not Mine!");
        exit();
    }
}
@cmdecho = 0;           // Turn off command echo
userbrk(0);           // Disable user break

wcen = @wheelcen;     // Read predefined centers,
p2 = @wh_crank;       // and Crank position
p1 = @pistonjoin;

@selmask = 0;         // Enable all entities
@snapflag |= 0x2000; // Enable Mask

set_mask(0,"",0);    // Mask off all layer
set_mask(0,"WHEEL",1); // On Layer WHEEL only
wheel = getesel(-2); // Select All ...

set_mask(0,"",0);    // Mask off all layer
set_mask(0,"CRANK",1); // On Layer CRANK only
crank = getesel(-2); // Select All ...

set_mask(0,"",0);    // Mask off all layer
set_mask(0,"PISTON",1); // On Layer PISTON only
piston = getesel(-2); // Select All ...

layer("WHEEL","", -1,1); // Turn off layers...
layer("PISTON","", -1,1);
layer("CRANK","", -1,1);
layer("FILLET","", -1,1);
command("ze");        // Zoom to Drawing Extent

keyin();              // Add for TwinCAD to start regeneration
regen_count();       // Reset regeneration count
redraw_count();      // Reset redraw count

set_mask(0,"",0);    // Clear all layer mask

r = p2,wcen;         // Calculate distance
b = p1,p2;
c = b/r;
c *= c;

set_map(0,wcen); plotent(wheel); set_map(-1);
set_map(1,p1); plotent(piston); set_map(-1);
set_map(2,p1); plotent(crank); set_map(-1);

crot = wrot = cos(0),sin(0);

setplot(0x8a,0,0,0);
set_map(-5,0,0);
for(v1=0;;v1+=division) {
    put_map(1,p1);
    set_map(-5,crot); put_map(2,p1);
    set_map(-5,wrot); put_map(0,wcen);
    v1 %= 360;
    wrot = cos(v1),sin(v1);
    p0 = r*(wrot.x+sqrt(c-wrot.y*wrot.y)),0;
    p2 = (p0-r*wrot)/b;
    p0 += wcen;
    if ( userbrk(-1) )

```

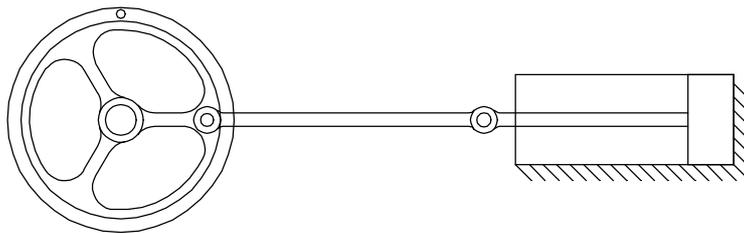
```

        break;
    //
    //   Add additional 'if' block to consider the case when
    //   the operator has scrolled the display or changed
    //   the size of the drawing area under Windows.
    //           Mar 01, 1997 by Kang, Hsin-Min
    //
    if ( identify("regen_count") ) {
refresh
        if ( regen_count() || redraw_count() ) { // Have screen
            redraw_count(); // Reset redraw count
            p1 = @pistonjoin; // Redefine map
            set_map(0,wcen); plotent(wheel); set_map(-1);
            set_map(1,p1); plotent(piston); set_map(-1);
            set_map(2,p1); plotent(crank); set_map(-1);
            crot = p2;
            p1 = p0;
            set_map(-5,0,0); // Reset to last state
            continue;
        }
    }

    set_map(-5,crot); put_map(0,wcen);
    set_map(-5,0,0); put_map(2,p1);
    crot = p2; put_map(1,p1);
    p1 = p0;
}
@cmdecho = 1;
userbrk(1);
layer("WHEEL","",-1,0); // Turn ON layers...
layer("PISTON","",-1,0);
layer("CRANK","",-1,0);
layer("FILLET","",-1,0);
exit();
}

```

PISTON/WHEEL DEMO



RUN PISTON.TCL AND SEE WHAT HAPPEN!!

PLOT2.TCL

The source program listed below is an example for plotting a 3-D function. The plotting result is shown below. The program code is provided here to show how to generate function plotting with TCL and specifically with the use of TCL preprocessor directives. The programming technique applied here is quite straightforward. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```

/*****
    Program:      PLOT2.TCL
    Author:       Kang, Hsin-Min
    Title:        Function Plotting
    Purpose:      Example of TCL Programming
*****/

#outmsg "\nGenerate plotting of surface(x,y) = cosr(sqrt(2*x*x+y*y))"
#ifyes "\nGenerate no real entities but plotting only (yes/no) (Y) ?"
#define newent(x)  plotent(x)
#endif
#define dx  v1
#define dy  v2
#define x   v3
#define y   v4
#define z1  v5
#define z2  v6

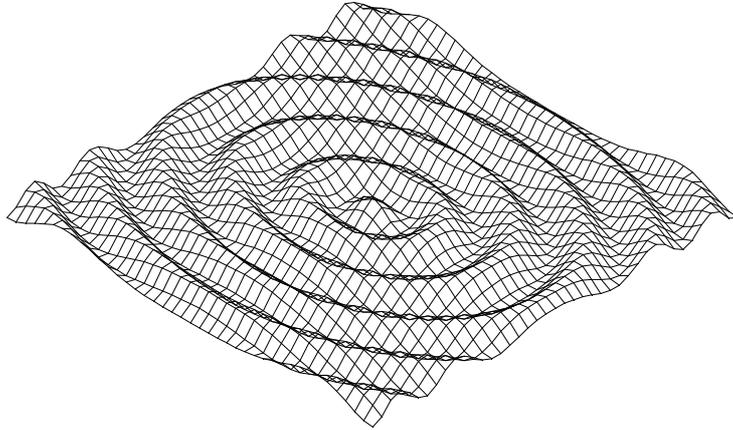
#define cs(x,y)  cosr(sqrt(2*(x)*(x)+(y)*(y)))

plot2()
{
//    double      dx, dy;
//    double      x, y, z1, z2;
    int  secho;

    secho = @cmdecho;
    @cmdecho = 0;
    command("zw -200,-200 200,200 vpoint 1,1,1");
    setplot(0x0f,0,1,0);
    dx = 1;      dy = 1;
    for(x=-20;x<=20;x+=dx) {
        p1 = p(x,-20)*10;
        z1 = cs(x,-20)*10;
        for(y=-20+dy;y<=20;y+=dy) {
            z2 = cs(x,y)*10;
            p2 = p(x,y)*10;
            l1 = p1,p2,z1,z2;
            newent(l1);
            p1 = p2;
            z1 = z2;
        }
    }
    for(y=-20;y<=20;y+=dx) {
        p1 = p(-20,y)*10;
        z1 = cs(-20,y)*10;
        for(x=-20+dx;x<=20;x+=dx) {
            z2 = cs(x,y)*10;
            p2 = p(x,y)*10;
            l1 = p1,p2,z1,z2;
            newent(l1);
            p1 = p2;
        }
    }
}

```

```
        z1 = z2;  
    }  
}  
s1="\nYou may use cursor key to rotate the view point...";  
command("vpoint 'printf(s1) d");  
@cmdecho=secho;  
}
```



POLYSTAR.TCL

The source program listed below is an interesting example for showing how to draw a star easily. The algorithm is generalized to draw a star with multiple vertices, as implied by the name of the program. The programming technique applied here is quite straightforward, once you get the algorithm figured out. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed. Interested readers are encouraged to modify the program and to add dragging features while generating the polystars.

```

/*****
    Program:      POLYSTAR.TCL
    Purpose:      Draw General PolyStar
    Author:       Kang, Hsin-min
    Copyright ©1994 By TCAM Development House, Taipei, R.O.C.

    This program shows how to draw a star in a most quick way. The algorithm has
    been generalized to draw stars with multiple vertices, as the name of the
    program implies. The reader is encouraged to rewrite the program and to add
    powerful dragging capability for the creation of a polystar or polystars,
    which is left as an exercise.
    *****/

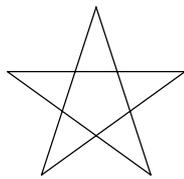
COMMAND PolyStar()
{
    POINT    center;        // Center of polystar
    double   radius;        // Radius of polystar
    double   angle;         // Starting angle
    double   vspan;         // Span between vertex
    int      vtexno;        // Number of Vertex
    int      stepno;        // Step Number
    int      cycleno;       // Cycle Number
    int      cmdecho;       // Used to save @cmdecho

    vtexno = getv("\nPolyStar -- Enter number of vertex (5): ",5);
    if ( vtexno <= 4 ) {
        printf("\nSorry, can't do that with a vertex number less than 5.");
        exit(0);
    }
    stepno = vtexno>>1;     // Calculate default step number
    if ( !(vtexno&1) )
        stepno-=1;
    stepno = getv(prints("\nPolyStar -- Enter number of vertex step
(%d): ",stepno),stepno);
    if ( stepno <= 1 ) {
        printf("\nSorry, can't do that with a step number < 2");
        exit(0);
    }
    if ( stepno >= vtexno ) {
        printf("\nSorry, can't do that with a step number >= vertex
number");
        exit(0);
    }
    center = getp("\nPolyStar -- Center of the polystar:");
    radius = getdist("\nPolyStar -- Radius of the polystar
(5)",10.,center);
    angle = getangle("\nPolyStar -- Starting Angle
(90):",90.,center);

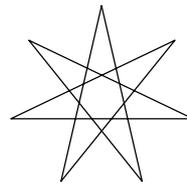
    vspan = 360./vtexno;

```

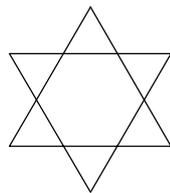
```
cycleno = gcd(vtexno,stepno);  
vtexno /= cycleno;  
stepno /= cycleno;  
cmdecho = @cmdecho;  
@cmdecho = 0;  
for(i0=0;i0<cycleno;i0++) {  
  V1 = angle+vspan*i0;  
  P1 = center+P(radius,(V1)A);  
  P2 = center+P(radius,(V1+vspan*stepno*cycleno)A);  
  command("LINE P1 P2 ^M");  
  command("ARRAY L P \@center \@vtexno \@stepno*360 Y");  
}  
@cmdecho = cmdecho;  
}
```



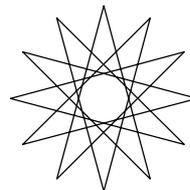
2/5 Star



3/7 Star



2/6 Star



5/12 Star

SPIRAL.TCL

The source program listed below shows an example of using **sarc()** to approximate a 2-D parametric function. The programming technique applied here is quite straightforward. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```

/*****
    Program:      SPIRAL.TCL
    Author:       Kang, Hsin-Min
    Purpose:      Demonstration of Parametric Function Approximation using sarc()
    Effect:       Plot of Spiral of Archimedes

                X(t)   = t*cos(t)
                Y(t)   = t*sin(t)
                X'(t)  = cos(t) - t*sin(t)
                  = cos(t) - Y(t)
                Y'(t)  = sin(t) + t*cos(t)
                  = sin(t) + X(t)
*****/

#outmsg "\nGenerate plotting of Spiral of Archimedes.."
#ifyes "\nGenerate no real entities but plotting only (yes/no) (Y) ?"
#define newent(x) plotent(x)
#endif
COMMAND Spiral()
{
    double      r, maxangle;
    double      dt, t;
    int         secho;

    maxangle = getv("\nEnter Maxium Angle in degree: ",720);

    maxangle = dtr(maxangle);

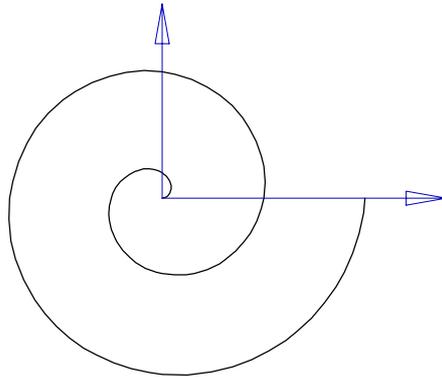
    secho = @cmdecho;
    @cmdecho = 0;
    r = maxangle*1.5;
    command("zw -\@r,-\@r \@r,\@r");
    setplot(0x09,0,1,0);
    dt = pi/4;
    p1 = 0,0;
    p2 = 1,0;

    for(t=dt;t<=maxangle;t+=dt) {
        p3.x = t*cosr(t);
        p3.y = t*sinr(t);
        p4.x = cosr(t) - t*sinr(t);
        p4.y = sinr(t) + t*cosr(t);
        sarc(p1,p2,p3,p4,&a1);
        newent(a1);
        newent(a2);
        p1 = p3;
        p2 = p4;
    }
}
/*
    setplot(0x0f,0,1,0);

    dt = pi/100;
    p1 = 0,0;

```

```
for(t=dt;t<=maxangle;t+=dt) {  
    p2.x = t*cosr(t);  
    p2.y = t*sinr(t);  
    l1 = p1,p2;  
    newent(l1);  
    p1 = p2;  
}  
*/  
@cmdecho=secho;  
}
```



XPDL.TCL

The source program listed below is an example for implementing a useful extended command which will create a Line entity to replace the extension line of a selected dimension without exploding it. This command is useful when you need to trim or break the extension lines and make the drawing easier to read. The program body is included with TwinCAD CD-ROM and can be found in TwinCAD\TCL folder when installed.

```

/*****
Command:      XPDL -- Explode Part of Dimension into Line

Purpose:      This program is used to replace the extension line of a
dimension entity with a real line for further trimming or breaking purpose.

Programmed by Kang, Hsin-Min.
Released in 1995 as a working example.
Revised on Feb 09, 1998 -- Accept line entity for breaking
Revised on Apr 22, 1998 Include Japanese Text in SJIS codes
Copyright(c) 1995, 1998 TCAM Development House, Taipei. All rights Reserved.

Status:       TwinCAD Standard Command
Comments:     The selected dimension's extension line nearest to the pick
point will be suppressed and replaced with a real line entity.
*****/

//#include "tclcad.h"
// Resolve definition from tclcad.h
#define T_LINE      2
#define T_DIM       9
#define FLINE      4
#define FDIM       0x200
#define DIMSE1     0x10 /* Suppress first extension line */
#define DIMSE2     0x20 /* Suppress second extension line */
#define OSINT      0x20 /* Intersection of two objects */
#define DIMLINE    0
#define DIMANG     1

#define LoopForever while(1)
#define nearp1(p1,p2,po) (v(p1,po)<=v(p2,po))

#define TGBIG5      1
#define TGGB2312   2
#define TGSJIS     5
#define TGKOREAN   3

char  szText[5][70];          // Our display text buffer
char  szAsciiText[][70]={    // Default English Display Text
/* 0 */  "\nXPDL -- Please select dimension by its extension line: ",
/* 1 */  "\n** Please select linear or angular dimension.",
/* 2 */  "\nPlease indicate where to break: ",
/* 3 */  " -- %d lines produced."
};
char  szBig5Text[][60]={     // Default Chinese Big-5 Display Text
/* 0 */  "\n請選取尺度標註圖元的延伸線: ",
/* 1 */  "\n** 請選取線性尺度標註或角度尺度標註圖元.",
/* 2 */  "\n請指示斷離點: ",
/* 3 */  " -- 有 %d 線產生."
};
char  szGB2312Text[][70]={   // Default Chinese GB2312 Display Text

```

```

/* 0 */  "\n 恁 喜僅梓蛸苧唵腔哇托盃: ",
/* 1 */  "\n** 恁 盃俶喜僅梓蛸麼褒僅喜僅梓蛸苧唵.",
/* 2 */  "\n 硤尢剿燭莢: ",
/* 3 */  " -- 蚺 %d 盃莉汜."
};
//
//    The following Japanese messages are translated by _____.
//
char  szSJISText[][70]={ // Default Japanese SJIS Display Text
/* 0 */  "\nXPDL --                : ",
/* 1 */  "\n**                ",
/* 2 */  "\n                ",
/* 3 */  " -- %d                "
};
//
//    The following Korean messages are translated by _____.
//
char  szKoreanText[][70]={ // Default Korean Display Text
/* 0 */  "\nXPDL -- Please select dimension by its extension line: ",
/* 1 */  "\n** Please select linear or angular dimension.",
/* 2 */  "\nPlease indicate where to break? ",
/* 3 */  " -- %d lines produced."
};

COMMAND XPDL()
{
    ENTDATA data;

    switch(@tgftype) {
        case TGBIG5:      setdata(szText,szBig5Text);    break;
        case TGGB2312:   setdata(szText,szGB2312Text);  break;
        case TGSJIS:     setdata(szText,szSJISText);    break;
        case TGKOREAN:   setdata(szText,szKoreanText);  break;
        default:         setdata(szText,szAsciiText);   break;
    }

    LoopForever {
        e1 = gete(szText[0],FDIM|FLINE);
        if ( userbrk(-1) ) break;
        if ( !ent_ok(e1) ) break;
        if ( ent_type(e1)==T_DIM ) {
            data = ent_read(e1);
            switch(data.dim.type&0x0f) {
                case DIMLINE: XpLinear();    continue;
                case DIMANG:  XpAngular();   continue;
            }
        }
        if ( ent_type(e1)==T_LINE ) {
            XpLine();
            continue;
        }
        printf(szText[1]);
    }
}

XpLinear()
{
    ENTDATA data;

    data = ent_read(e1);
}

```

```
if ( nearpl(data.dim.pds,data.dim.pde,e1.pick) ) {
    if ( data.dim.flag & DIMSE1 )
        return; // First Extension is already suppressed!
    data.dim.flag |= DIMSE1;
    l1.ps = data.dim.ps;
    l1.pe = data.dim.pds;
}
else {
    if ( data.dim.flag & DIMSE2 )
        return; // Second Extension is already suppressed!
    data.dim.flag |= DIMSE2;
    l1.ps = data.dim.pe;
    l1.pe = data.dim.pde;
}

@osmode = OSINT;
p1 = getp(szText[2],l1.ps);
@osmode = 0;

ent_write(e1,data);

l1.pe = l1,V(l1)+@dimexe;
l1.ps = l1,@dimexo;

CreateLine();
}

XpAngular()
{
    ENTDATA data;
    double len;

    data = ent_read(e1);

    l1.ps = data.dim.ps;
    if ( nearpl(data.dim.pds,data.dim.pde,e1.pick) ) {
        if ( data.dim.flag & DIMSE1 )
            return; // First Extension is already suppressed!
        data.dim.flag |= DIMSE1;
        l1.pe = data.dim.pds;
        len = data.dim.pe.x;
    }
    else {
        if ( data.dim.flag & DIMSE2 )
            return; // Second Extension is already suppressed!
        data.dim.flag |= DIMSE2;
        l1.pe = data.dim.pde;
        len = data.dim.pe.y;
    }

    @osmode = OSINT;
    p1 = getp(szText[2],l1.ps);
    @osmode = 0;

    ent_write(e1,data);

    if ( len < V(data.dim.ps,l1.pe) ) {
        l1.pe = l1,V(l1)+@dimexe;
        l1.ps = l1,len+@dimexo;
    }
}
```

```

        else {
            l1.pe = l1,V(l1)-@dimexe;
            l1.ps = l1,len-@dimexo;
        }

        CreateLine();
    }
    /*
    L1 -- New line to create
    P1 -- Point to split the line
    */
CreateLine()
{
    ENTDATA data, data1;

    data = ent_read(e1);
    p1 = p1,l1;
    if ( (p1==l1.ps) || (p1==l1.pe) || !in_span(l1,p1) ) {
        e2=newent(l1);
        data1 = ent_read(e2);
        data1.layer = data.layer;
        data1.color = data.color;
        data1.ltypeno = data.ltypeno;
        ent_write(e2,data1);
        printf(szText[3],1);
    }
    else {
        l2.pe = L(p1,l1.ps),@dingap;
        l2.ps = l1.ps;
        l1.ps = L(p1,l1.pe),@dingap;

        e2=newent(l1);
        e3=newent(l2);

        data1 = ent_read(e2);
        data1.layer = data.layer;
        data1.color = data.color;
        data1.ltypeno = data.ltypeno;
        ent_write(e2,data1);

        data1 = ent_read(e3);
        data1.layer = data.layer;
        data1.color = data.color;
        data1.ltypeno = data.ltypeno;
        ent_write(e3,data1);
        printf(szText[3],2);
    }
}

XpLine()
{
    ENTDATA data, data1;
    l1 = e1;
    @osmode = OSINT;
    p1 = getp(szText[2],l1.ps);
    @osmode = 0;
    data = ent_read(e1);
    p1 = p1,l1;
    if ( (p1==l1.ps) || (p1==l1.pe) || !in_span(l1,p1) ) {
        return;
    }
}

```

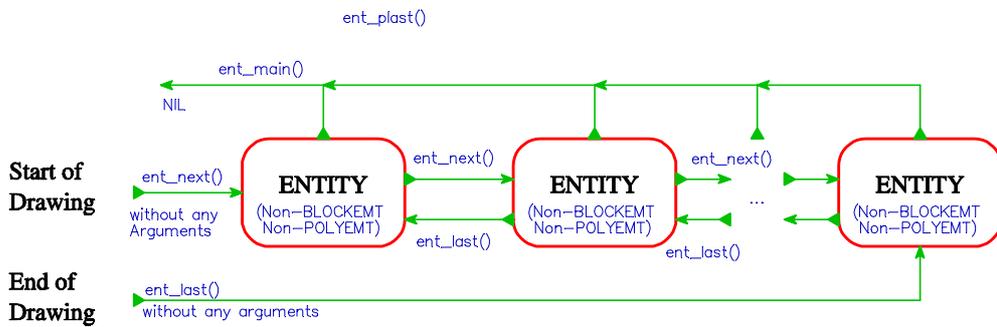
```
    else {
      l2.pe = L(p1,l1.ps),@dimgap;
      l2.ps = l1.ps;
      l1.ps = L(p1,l1.pe),@dimgap;

      data.line.ps = l1.ps;
      data.line.pe = l1.pe;
      ent_write(e1,data);
      e2=newent(l2);
      data1 = ent_read(e2);
      data1.layer = data.layer;
      data1.color = data.color;
      data1.ltypeno = data.ltypeno;
      ent_write(e2,data1);
    }
  }
```

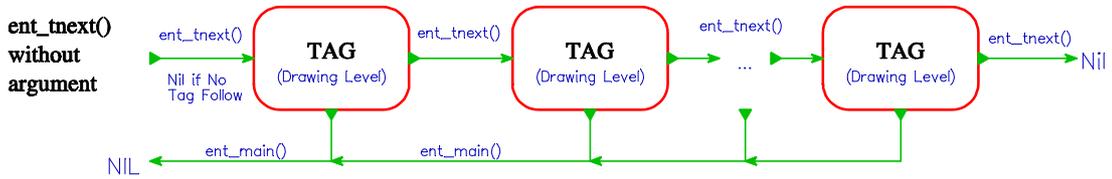
Appendix

Map for Drawing Database Traveling

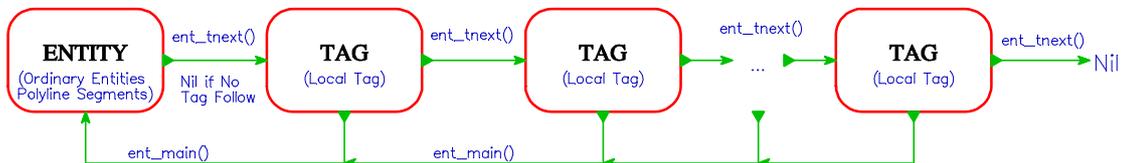
Top Level Drawing Entities Traveling



Traveling Tags of Drawing Level



Geometry Map for Tagged Entities



Geometry Map for INSERTs

