

Free Pascal supplied units :
Reference guide.

Reference guide for standard Free Pascal units.

1.9

April 2002

Michaël Van Canneyt
Florian Klämpfl

Contents

1	The CRT unit.	28
1.1	Types, Variables, Constants	28
1.2	Procedures and Functions	29
	AssignCrt	29
	CursorBig	30
	ClrEol	30
	ClrScr	30
	CursorOff	31
	CursorOn	31
	Delay	31
	DelLine	32
	GotoXY	32
	HighVideo	33
	InsLine	33
	KeyPressed	34
	LowVideo	34
	NormVideo	35
	NoSound	35
	ReadKey	35
	Sound	36
	TextBackground	36
	TextColor	37
	TextMode	37
	WhereX	37
	WhereY	38
	Window	38
2	The DOS unit.	40
2.1	Types, Variables, Constants	40
	Constants	40
	File attributes	40

	fmXXXX	40
	Other	41
	Types	41
	Variables	43
2.2	Function list by category	43
	File handling	43
	Directory and disk handling	44
	Process handling	44
	System information	44
2.3	Functions and Procedures	45
	AddDisk	45
	DiskFree	45
	DiskSize	46
	DosExitCode	46
	DosVersion	47
	EnvCount	48
	EnvStr	48
	Exec	48
	FExpand	48
	FindClose	49
	FindFirst	49
	FindNext	50
	FSearch	50
	FSplit	51
	GetCBreak	51
	GetDate	52
	GetEnv	52
	GetFAttr	53
	GetFTime	53
	GetIntVec	54
	GetLongName	54
	GetShortName	55
	GetTime	55
	GetVerify	56
	Intr	56
	Keep	56
	MSDos	56
	PackTime	57
	SetCBreak	57
	SetDate	58

SetFAttr	58
SetFTime	58
SetIntVec	59
SetTime	59
SetVerify	59
SwapVectors	59
UnPackTime	60
3 The DXELOAD unit	61
3.1 Introduction	61
3.2 Constants, types and variables	61
Constants	61
Types	61
3.3 Functions and Procedures	61
dx_load	61
4 The EMU387 unit	63
4.1 Functions and procedures	63
npxsetup	63
5 The GETOPTS unit.	64
5.1 Types, Constants and variables :	64
Constants	64
Types	64
Variables	65
5.2 Procedures and functions	65
GetLongOpts	65
Getopt	65
6 The GPM unit	68
6.1 Introduction	68
6.2 Constants, types and variables	68
constants	68
Types	69
Variables	70
6.3 Functions and procedures	70
Gpm_AnyDouble	70
Gpm_AnySingle	71
Gpm_AnyTriple	71
Gpm_Close	71
Gpm_FitValues	71

Gpm_FitValuesM	71
Gpm_GetEvent	72
Gpm_GetLibVersion	73
Gpm_GetServerVersion	73
Gpm_GetSnapshot	73
Gpm_LowerRoi	73
Gpm_Open	74
Gpm_PopRoi	74
Gpm_PushRoi	74
Gpm_RaiseRoi	74
Gpm_Repeat	75
Gpm_StrictDouble	75
Gpm_StrictSingle	75
Gpm_StrictTriple	75
7 The GO32 unit	76
7.1 Introduction	76
7.2 Protected mode memory organization	76
What is DPMI	76
Selectors and descriptors	76
FPC specialities	77
DOS memory access	77
I/O port access	77
Processor access	77
Interrupt redirection	77
Handling interrupts with DPMI	78
Protected mode interrupts vs. Real mode interrupts	78
Creating own interrupt handlers	78
Disabling interrupts	78
Hardware interrupts	78
Software interrupts	80
Real mode callbacks	82
7.3 Types, Variables and Constants	83
Constants	83
Constants returned by get_run_mode	83
Processor flags constants	83
Predefined types	83
Variables.	84
7.4 Functions and Procedures	85
allocate_ldt_descriptors	85

allocate_memory_block	87
copyfromdos	87
copytodos	87
create_code_segment_alias_descriptor	88
disable	88
dosmemfillchar	88
dosmemfillword	89
dosmemget	89
dosmemmove	90
dosmempu	90
enable	90
free_ldt_descriptor	91
free_memory_block	91
free_rm_callback	91
get_cs	92
get_descriptor_access_rights	92
get_ds	92
get_linear_addr	92
get_meminfo	93
get_next_selector_increment_value	94
get_page_size	94
get_pm_interrupt	94
get_rm_callback	95
get_rm_interrupt	97
get_run_mode	98
get_segment_base_address	98
get_segment_limit	99
get_ss	99
global_dos_alloc	99
global_dos_free	101
inportb	101
inportl	101
inportw	101
lock_code	102
lock_data	102
lock_linear_region	102
outportb	103
outportl	103
outportw	103
realintr	104

seg_fillchar	104
seg_fillword	105
segment_to_descriptor	105
seg_move	106
set_descriptor_access_rights	106
set_pm_interrupt	106
set_rm_interrupt	107
set_segment_base_address	108
set_segment_limit	108
tb_size	108
transfer_buffer	109
unlock_code	109
unlock_data	109
unlock_linear_region	109
8 The GRAPH unit.	110
8.1 Introduction	110
Requirements	110
A word about mode selection	110
8.2 Constants, Types and Variables	115
Types	115
8.3 Function list by category	115
Initialization	115
screen management	116
Color management	116
Drawing primitives	117
Filled drawings	117
Text and font handling	118
8.4 Functions and procedures	118
Arc	118
Bar	118
Bar3D	118
Circle	119
ClearDevice	119
ClearViewPort	119
CloseGraph	119
DetectGraph	119
DrawPoly	120
Ellipse	120
FillEllipse	120

FillPoly	120
FloodFill	120
GetArcCoords	121
GetAspectRatio	121
GetBkColor	121
GetColor	121
GetDefaultPalette	121
GetDriverName	121
GetFillPattern	122
GetFillSettings	122
GetGraphMode	122
GetImage	122
GetLineSettings	122
GetMaxColor	122
GetMaxMode	123
GetMaxX	123
GetMaxY	123
GetModeName	123
GetModeRange	123
GetPalette	124
GetPaletteSize	124
GetPixel	124
GetTextSettings	124
GetViewSettings	124
GetX	124
GetY	125
GraphDefaults	125
GraphErrorMsg	125
GraphResult	125
ImageSize	126
InitGraph	126
InstallUserDriver	126
InstallUserFont	127
Line	127
LineRel	127
LineTo	127
MoveRel	127
MoveTo	128
OutText	128
OutTextXY	128

PieSlice	128
PutImage	128
PutPixel	129
Rectangle	129
RegisterBGIDriver	129
RegisterBGIFont	129
RestoreCRTMode	129
Sector	130
SetActivePage	130
SetAllPalette	130
SetAspectRatio	130
SetBkColor	130
SetColor	131
SetFillPattern	131
SetFillStyle	131
SetGraphBufSize	131
SetGraphMode	132
SetLineStyle	132
SetPalette	132
SetRGBPalette	132
SetTextJustify	133
SetTextStyle	133
SetUserCharSize	133
SetViewPort	134
SetVisualPage	134
SetWriteMode	134
TextHeight	134
TextWidth	134
8.5 Target specific issues	135
DOS	135
WINDOWS	135
LINUX	135
9 The HEAPTRC unit.	136
9.1 Purpose	136
9.2 Usage	136
9.3 Constants, Types and variables	137
9.4 Functions and procedures	138
DumpHeap	138
MarkHeap	138

SetExtraInfo	139
SetHeapTraceOutput	140
10 The IPC unit.	141
10.1 Types, Constants and variables :	141
Variables	141
Constants	141
Types	142
10.2 Functions and procedures	146
ftok	146
msgget	146
msgsnd	146
msgrcv	147
msgctl	147
semget	150
semop	150
semctl	151
shmget	155
shmat	156
shmdt	156
shmctl	156
11 The KEYBOARD unit	159
11.1 Constants, Type and variables	159
Constants	159
Types	161
11.2 Functions and Procedures	162
DoneKeyboard	162
FunctionKeyName	162
GetKeyboardDriver	163
GetKeyEvent	163
GetKeyEventChar	164
GetKeyEventCode	164
GetKeyEventFlags	165
GetKeyEventShiftState	165
GetKeyEventUniCode	166
InitKeyBoard	166
IsFunctionKey	166
KeyEventToString	167
PollKeyEvent	167
PollShiftStateEvent	168

PutKeyEvent	168
SetKeyboardDriver	169
ShiftStateToString	170
TranslateKeyEvent	170
TranslateKeyEventUniCode	170
11.3 Keyboard scan codes	170
11.4 Writing a keyboard driver	171
12 The LINUX unit.	177
12.1 Type, Variable and Constant declarations	177
Types	177
Variables	180
Constants	181
12.2 Function list by category	184
File Input/Output routines	184
General File handling routines	185
Pipes, FIFOs and streams	186
Directory handling routines	186
Process handling	186
Signals	187
System information	187
Terminal functions	188
Port input/output	188
Utility routines	188
12.3 Functions and procedures	189
Access	189
Alarm	190
AssignPipe	190
AssignStream	191
BaseName	193
CFMakeRaw	193
CFSetISpeed	193
CFSetOSpeed	194
Chown	194
Chmod	195
Clone	196
CloseDir	198
CreateShellArgV	198
DirName	199
Dup	199

Dup2	200
EpochToLocal	201
Execl	201
Execle	202
Execlp	203
Execv	203
Execve	204
Execvp	205
FD_ZERO	206
FD_Clr	206
FD_IsSet	206
FD_Set	207
fdClose	207
fdFlush	207
fdOpen	207
fdRead	208
fdSeek	210
fdTruncate	210
fdWrite	210
FExpand	210
FLock	211
FNMatch	211
FSearch	212
FSplit	212
FSSStat	213
FStat	214
Fcntl	215
Fcntl	215
Fork	216
FRename	216
GetDate	217
GetDateTime	217
GetDomainName	217
GetEGid	218
GetEUid	218
GetEnv	219
GetEpochTime	219
GetFS	219
GetGid	220
GetHostName	220

GetLocalTimezone	221
GetPid	221
GetPPid	221
GetPriority	222
GetTime	222
GetTimeOfDay	223
GetTimeOfDay	223
GetTimezoneFile	223
GetUid	224
Glob	224
GlobFree	225
IOCtl	225
IOperm	225
IsATTY	226
S_ISBLK	226
S_ISCHR	226
S_ISDIR	226
S_ISFIFO	226
S_ISLNK	227
S_ISREG	227
S_ISSOCK	227
Kill	228
LStat	228
Link	229
LocalToEpoch	230
MkFifo	231
MMap	231
MUnMap	233
Nice	233
Octal	234
OpenDir	234
pause	235
PClose	235
POpen	236
ReadDir	236
ReadLink	237
ReadPort	238
ReadPortB	238
ReadPortL	238
ReadPortW	239

ReadTimezoneFile	239
SeekDir	239
Select	240
SelectText	241
SetPriority	241
Shell	241
SigAction	242
SigPending	243
SigProcMask	243
SigRaise	244
SigSuspend	244
Signal	245
StringToPPchar	245
SymLink	246
SysInfo	247
TCDrain	248
TCFlow	248
TCFlush	249
TCGetAttr	249
TCGetPGrp	250
TCSendBreak	250
TCSetAttr	250
TCSetPGrp	251
TTYName	251
TellDir	251
Umask	251
Uname	252
UnLink	252
Utime	252
WaitPid	253
WritePort	254
WritePortB	254
WritePortL	254
WritePortW	255
13 The MATH unit	256
13.1 Constants and types	256
13.2 Function list by category	257
Min/max determination	257
Angle conversion	257

Trigonometric functions	257
Hyperbolic functions	258
Exponential and logarithmic functions	258
Number converting	258
Statistical functions	258
Geometrical functions	259
13.3 Functions and Procedures	259
arccos	259
arcosh	259
arcsin	260
arctan2	261
arsinh	261
artanh	262
ceil	262
cosh	263
cotan	263
cycletorad	263
degtograd	264
degtorad	264
floor	265
frexp	265
gradtodeg	266
gradtorad	266
hypot	267
intpower	267
ldexp	268
lnxpl	268
log10	269
log2	269
logn	270
max	270
maxIntValue	271
maxvalue	271
mean	272
meanandstddev	273
min	273
minIntValue	274
minvalue	275
momentskewkurtosis	275
norm	276

popnstddev	277
popnvariance	277
power	278
radtocycle	279
radtodeg	279
radtograd	280
randg	280
sincos	281
sinh	281
stddev	282
sum	282
sumofsquares	283
sumsandsquares	284
tan	284
tanh	285
totalvariance	285
variance	286
14 The MMX unit	288
14.1 Variables, Types and constants	288
14.2 Functions and Procedures	289
Emms	289
15 The MOUSE unit	290
15.1 Constants, Types and Variables	290
Constants	290
Types	290
Variables	291
15.2 Functions and procedures	291
DetectMouse	291
DoneMouse	292
GetMouseButtons	292
GetMouseDriver	293
GetMouseEvent	293
GetMouseX	293
GetMouseY	294
HideMouse	294
InitMouse	295
PollMouseEvent	295
PutMouseEvent	295
SetMouseDriver	296

SetMouseXY	296
ShowMouse	296
15.3 Writing a custom mouse driver	297
16 The MsMouse unit	300
16.1 Constants, types and variables	300
16.2 Functions and procedures	301
GetLastButtonPress	301
GetLastButtonRelease	302
GetMouseState	303
HideMouse	304
InitMouse	304
LPressed	305
MPressed	305
RPressed	305
SetMouseAscii	305
SetMouseHideWindow	306
SetMousePos	307
SetMouseShape	308
SetMouseSpeed	309
SetMouseWindow	310
SetMouseXRange	310
SetMouseYRange	311
ShowMouse	311
17 The Objects unit.	312
17.1 Constants	312
17.2 Types	313
17.3 Procedures and Functions	314
NewStr	314
DisposeStr	315
Abstract	315
RegisterObjects	315
RegisterType	315
LongMul	317
LongDiv	317
17.4 TRect	317
TRect.Empty	318
TRect.Equals	318
TRect.Contains	319
TRect.Copy	319

TRect.Union	319
TRect.Intersect	320
TRect.Move	321
TRect.Grow	321
TRect.Assign	322
17.5 TObject	322
TObject.Init	322
TObject.Free	322
TObject.Done	323
17.6 TStream	324
TStream.Get	324
TStream.StrRead	325
TStream.GetPos	325
TStream.GetSize	326
TStream.ReadStr	326
TStream.Open	327
TStream.Close	327
TStream.Reset	328
TStream.Flush	328
TStream.Truncate	328
TStream.Put	328
TStream.StrWrite	329
TStream.WriteStr	329
TStream.Seek	329
TStream.Error	329
TStream.Read	330
TStream.Write	330
TStream.CopyFrom	330
17.7 TDosStream	331
TDosStream.Init	332
TDosStream.Done	332
TDosStream.Close	332
TDosStream.Truncate	333
TDosStream.Seek	333
TDosStream.Open	334
TDosStream.Read	335
TDosStream.Write	335
17.8 TBufStream	335
TBufStream.Init	336
TBufStream.Done	336

TBufStream.Close	337
TBufStream.Flush	337
TBufStream.Truncate	338
TBufStream.Seek	338
TBufStream.Open	338
TBufStream.Read	338
TBufStream.Write	339
17.9 TMemoryStream	339
TMemoryStream.Init	339
TMemoryStream.Done	340
TMemoryStream.Truncate	340
TMemoryStream.Read	340
TMemoryStream.Write	341
17.10 TCollection	341
TCollection.Init	342
TCollection.Load	342
TCollection.Done	343
TCollection.At	343
TCollection.IndexOf	344
TCollection.GetItem	344
TCollection.LastThat	345
TCollection.FirstThat	345
TCollection.Pack	346
TCollection.FreeAll	347
TCollection.DeleteAll	348
TCollection.Free	348
TCollection.Insert	349
TCollection.Delete	349
TCollection.AtFree	350
TCollection.FreeItem	351
TCollection.AtDelete	351
TCollection.ForEach	352
TCollection.SetLimit	353
TCollection.Error	353
TCollection.AtPut	353
TCollection.AtInsert	353
TCollection.Store	354
TCollection.PutItem	354
17.11 TSortedCollection	355
TSortedCollection.Init	356

TSortedCollection.Load	356
TSortedCollection.KeyOf	356
TSortedCollection.IndexOf	356
TSortedCollection.Compare	357
TSortedCollection.Search	357
TSortedCollection.Insert	358
TSortedCollection.Store	359
17.12 TStringCollection	360
TStringCollection.GetItem	360
TStringCollection.Compare	360
TStringCollection.FreeItem	361
TStringCollection.PutItem	361
17.13 TStrCollection	361
TStrCollection.GetItem	362
TStrCollection.Compare	362
TStrCollection.FreeItem	363
TStrCollection.PutItem	363
17.14 TUnSortedStrCollection	363
TUnSortedStrCollection.Insert	363
17.15 TResourceCollection	364
TResourceCollection.KeyOf	365
TResourceCollection.GetItem	365
TResourceCollection.FreeItem	365
TResourceCollection.PutItem	365
17.16 TResourceFile	366
TResourceFile Fields	366
TResourceFile.Init	366
TResourceFile.Done	366
TResourceFile.Count	367
TResourceFile.KeyAt	367
TResourceFile.Get	367
TResourceFile.SwitchTo	367
TResourceFile.Flush	367
TResourceFile.Delete	368
TResourceFile.Put	368
17.17 TStringList	368
TStringList.Load	368
TStringList.Done	369
TStringList.Get	369
17.18 TStrListMaker	369

TStrListMaker.Init	369
TStrListMaker.Done	370
TStrListMaker.Put	370
TStrListMaker.Store	370
18 The PORTS unit	371
18.1 Introduction	371
18.2 Types,constants and variables	371
Types	371
variables	372
19 The PRINTER unit.	373
19.1 Types, Constants and variables :	373
19.2 Procedures and functions	373
AssignLst	373
20 The SOCKETS unit.	375
20.1 Types, Constants and variables :	375
20.2 Functions and Procedures	376
Accept	376
Accept	377
Accept	378
Accept	378
Bind	378
Bind	379
Connect	379
Connect	380
Connect	380
Connect	380
GetPeerName	381
GetSocketName	381
GetSocketOptions	382
Listen	382
Recv	383
Send	383
SetSocketOptions	384
Shutdown	384
Sock2File	384
Sock2Text	385
Socket	385
SocketPair	385

Str2UnixSockAddr	385
21 The STRINGS unit.	386
21.1 Functions and procedures.	386
StrAlloc	386
StrCat	386
StrComp	387
StrCopy	387
StrDispose	388
StrECopy	388
StrEnd	389
StrIComp	389
StrLCat	390
StrLComp	390
StrLCopy	391
StrLen	391
StrLIComp	392
StrLower	392
StrMove	392
StrNew	393
StrPas	393
StrPCopy	394
StrPos	394
StrRScan	395
StrScan	395
StrUpper	395
22 The SYSUTILS unit.	397
22.1 Constants and types	397
22.2 Function list by category	400
String functions	400
Formatting strings	401
File input/output routines	402
File handling routines	402
Date/time routines	403
22.3 Miscellaneous conversion routines	404
22.4 Date and time functions	404
Date and time formatting characters	404
TDateTime	405
Date	405
DateTimeToFileDate	406

DateTimeToStr	406
DateTimeToString	407
DateTimeToSystemTime	408
DateTimeToTimeStamp	408
DateToStr	409
DayOfWeek	409
DecodeDate	409
DecodeTime	410
EncodeDate	410
EncodeTime	411
FileDateToDateTime	411
FormatDateTime	412
IncMonth	412
IsLeapYear	413
MSecsToTimeStamp	414
Now	414
StrToDate	415
StrToDateTime	415
StrToTime	416
SystemTimeToDateTime	417
Time	417
TimeStampToDateTime	418
TimeStampToMSecs	418
TimeToStr	418
22.5 Disk functions	419
AddDisk (Linux only)	419
CreateDir	419
DiskFree	420
DiskSize	420
GetCurrentDir	421
RemoveDir	421
SetCurrentDir	422
22.6 File handling functions	422
ChangeFileExt	422
DeleteFile	422
DoDirSeparators	423
ExpandFileName	423
ExpandUNCFileName	424
ExtractFileDir	424
ExtractFileDrive	425

ExtractFileExt	425
ExtractFileName	425
ExtractFilePath	426
ExtractRelativePath	426
FileAge	427
FileClose	427
FileCreate	427
FileExists	428
FileGetAttr	429
FileGetDate	430
FileOpen	430
FileRead	431
FileSearch	431
FileSeek	432
FileSetAttr (Not on Linux)	432
FileSetDate (Not on Linux)	433
FileTruncate	433
FileWrite	433
FindClose	433
FindFirst	434
FindNext	434
GetDirs	435
RenameFile	435
SetDirSeparators	436
22.7 PChar functions	436
Introduction	436
StrAlloc	437
StrBufSize	437
StrDispose	438
StrPCopy	438
StrPLCopy	438
StrPas	439
22.8 String handling functions	439
AdjustLineBreaks	439
AnsiCompareStr	439
AnsiCompareText	440
AnsiExtractQuotedStr	441
AnsiLastChar	442
AnsiLowerCase	442
AnsiQuotedStr	443

AnsiStrComp	443
AnsiStrIComp	444
AnsiStrLastChar	445
AnsiStrLComp	445
AnsiStrLIComp	446
AnsiStrLower	447
AnsiStrUpper	448
AnsiUpperCase	448
AppendStr	449
AssignStr	449
BCDToInt	450
CompareMem	450
CompareStr	451
CompareText	451
DisposeStr	452
FloatToStr	453
FloatToStrF	453
FloatToText	455
FmtStr	456
Format	456
FormatBuf	461
IntToHex	462
IntToStr	462
IsValidIdent	463
LastDelimiter	464
LeftStr	464
LoadStr	464
LowerCase	465
NewStr	465
QuotedStr	465
RightStr	466
StrFmt	466
StrLFmt	467
StrToInt	467
StrToIntDef	468
Trim	468
TrimLeft	469
TrimRight	470
UpperCase	470

23 The TYPINFO unit	472
23.1 Constants, Types and variables	472
Constants	472
types	472
23.2 Function list by category	475
Examining published property information	475
Getting or setting property values	476
Auxiliary functions	476
23.3 Functions and Procedures	477
FindPropInfo	477
GetEnumName	478
GetEnumProp	478
GetEnumValue	479
GetFloatProp	479
GetInt64Prop	480
GetMethodProp	481
GetObjectProp	483
GetObjectPropClass	484
GetOrdProp	484
GetPropInfo	485
GetPropInfos	486
GetPropList	487
GetPropValue	488
GetSetProp	488
GetStrProp	489
GetTypeData	490
GetVariantProp	490
IsPublishedProp	490
IsStoredProp	491
PropIsType	492
PropType	493
SetEnumProp	494
SetFloatProp	494
SetInt64Prop	494
SetMethodProp	495
SetObjectProp	495
SetOrdProp	495
SetPropValue	496
SetSetProp	496
SetStrProp	497

SetToString	497
SetVariantProp	498
StringToSet	498
24 The VIDEO unit	499
24.1 Constants, Type and variables	500
Constants	500
Types	501
Variables	502
24.2 Functions and Procedures	502
ClearScreen	503
DefaultErrorHandler	504
DoneVideo	504
GetCapabilities	504
GetCursorType	505
GetLockScreenCount	506
GetVideoDriver	507
GetVideoMode	507
GetVideoModeCount	508
GetVideoModeData	509
InitVideo	509
LockScreenUpdate	509
SetCursorPos	510
SetCursorType	511
SetVideoDriver	511
SetVideoMode	511
UnlockScreenUpdate	512
UpdateScreen	512
24.3 Writing a custom video driver	513

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the units that come standard with Free Pascal.

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures have their own subsections, and for each function or procedure we have the following topics:

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

The cross-references come in two flavors:

- References to other functions in this manual. In the printed copy, a number will appear after this reference. It refers to the page where this function is explained. In the on-line help pages, this is a hyperlink, on which you can click to jump to the declaration.
- References to Unix manual pages. (For Linux related things only) they are printed in `type-writer` font, and the number after it is the Unix manual section.

The chapters are ordered alphabetically. The functions and procedures in most cases also, but don't count on it. Use the table of contents for quick lookup.

Chapter 1

The CRT unit.

This chapter describes the CRT unit for Free Pascal, both under DOS LINUX and WINDOWS. The unit was first written for DOS by Florian klämpfl. The unit was ported to LINUX by Mark May¹, and enhanced by Michaël Van Canneyt and Peter Vreman. It works on the LINUX console, and in xterm and rxvt windows under X-Windows. The functionality for both is the same, except that under LINUX the use of an early implementation (versions 0.9.1 and earlier of the compiler) the crt unit automatically cleared the screen at program startup. This chapter is divided in two sections.

- The first section lists the pre-defined constants, types and variables.
- The second section describes the functions which appear in the interface part of the CRT unit.

1.1 Types, Variables, Constants

Color definitions :

```
Black = 0;
Blue = 1;
Green = 2;
Cyan = 3;
Red = 4;
Magenta = 5;
Brown = 6;
LightGray = 7;
DarkGray = 8;
LightBlue = 9;
LightGreen = 10;
LightCyan = 11;
LightRed = 12;
LightMagenta = 13;
Yellow = 14;
White = 15;
Blink = 128;
```

Miscellaneous constants

```
TextAttr: Byte = $07;
```

¹Current e-mail address mmay@dnaco.net

```
TextChar: Char = ' ';
CheckBreak: Boolean = True;
CheckEOF: Boolean = False;
CheckSnow: Boolean = False;
DirectVideo: Boolean = False;
LastMode: Word = 3;
WindMin: Word = $0;
WindMax: Word = $184f;
ScreenWidth = 80;
ScreenHeight = 25;
```

Some variables for compatibility with Turbo Pascal. However, they're not used by Free Pascal.

```
var
  checkbreak : boolean;
  checkeof : boolean;
  checksnow : boolean;
```

The following constants define screen modes on a DOS system:

```
Const
  bw40 = 0;
  co40 = 1;
  bw80 = 2;
  co80 = 3;
  mono = 7;
```

The `TextAttr` variable controls the attributes with which characters are written to screen.

```
var TextAttr : byte;
```

The `DirectVideo` variable controls the writing to the screen. If it is `True`, the the cursor is set via direct port access. If `False`, then the BIOS is used. This is defined under DOS only.

```
var DirectVideo : Boolean;
```

The `Lastmode` variable tells you which mode was last selected for the screen. It is defined on DOS only.

```
var lastmode : Word;
```

1.2 Procedures and Functions

AssignCrt

Declaration: `Procedure AssignCrt (Var F: Text);`

Description: `AssignCrt` Assigns a file `F` to the console. Everything written to the file `F` goes to the console instead. If the console contains a window, everything is written to the window instead.

Errors: None.

See also: [Window \(38\)](#)

Listing: crtex/ex1.pp

```
Program Example1;  
uses Crt;  
  
{ Program to demonstrate the AssignCrt function. }  
  
var  
  F : Text;  
begin  
  AssignCrt(F);  
  Rewrite(F); { Don't forget to open for output! }  
  WriteLn(F, 'This is written to the Assigned File');  
  Close(F);  
end.
```

CursorBig

Declaration: `Procedure CursorBig ;`

Description: Makes the cursor a big rectangle. Not implemented on LINUX.

Errors: None.

See also: [CursorOn \(31\)](#), [CursorOff \(31\)](#)

ClrEol

Declaration: `Procedure ClrEol ;`

Description: ClrEol clears the current line, starting from the cursor position, to the end of the window. The cursor doesn't move

Errors: None.

See also: [DelLine \(32\)](#), [InsLine \(33\)](#), [ClrScr \(30\)](#)

Listing: crtex/ex9.pp

```
Program Example9;  
uses Crt;  
  
{ Program to demonstrate the ClrEol function. }  
  
begin  
  Write('This line will be cleared from the',  
        ' cursor position until the right of the screen');  
  GotoXY(27,WhereY);  
  ReadKey;  
  ClrEol;  
  WriteLn;  
end.
```

ClrScr

Declaration: `Procedure ClrScr ;`

Description: `ClrScr` clears the current window (using the current colors), and sets the cursor in the top left corner of the current window.

Errors: None.

See also: `Window` ([38](#))

Listing: `crtex/ex8.pp`

```
Program Example8;  
uses Crt;  
  
  { Program to demonstrate the ClrScr function. }  
  
begin  
  Writeln('Press any key to clear the screen');  
  ReadKey;  
  ClrScr;  
  Writeln('Have fun with the cleared screen');  
end.
```

CursorOff

Declaration: `Procedure CursorOff ;`

Description: Switches the cursor off (i.e. the cursor is no longer visible). Not implemented on LINUX.

Errors: None.

See also: `CursorOn` ([31](#)), `CursorBig` ([30](#))

CursorOn

Declaration: `Procedure CursorOn ;`

Description: Switches the cursor on. Not implemented on LINUX.

Errors: None.

See also: `CursorBig` ([30](#)), `CursorOff` ([31](#))

Delay

Declaration: `Procedure Delay (DTime: Word);`

Description: `Delay` waits a specified number of milliseconds. The number of specified seconds is an approximation, and may be off a lot, if system load is high.

Errors: None

See also: `Sound` ([36](#)), `NoSound` ([35](#))

Listing: `crtex/ex15.pp`

```
Program Example15;  
uses Crt;  
  
{ Program to demonstrate the Delay function. }  
var  
    i : longint;  
begin  
    WriteLn(' Counting Down' );  
    for i:=10 downto 1 do  
        begin  
            WriteLn(i);  
            Delay(1000); {Wait one second}  
        end;  
    WriteLn('BOOM!!!' );  
end.
```

DelLine

Declaration: `Procedure DelLine ;`

Description: DelLine removes the current line. Lines following the current line are scrolled 1 line up, and an empty line is inserted at the bottom of the current window. The cursor doesn't move.

Errors: None.

See also: [ClrEol \(30\)](#), [InsLine \(33\)](#), [ClrScr \(30\)](#)

Listing: crtex/ex11.pp

```
Program Example10;  
uses Crt;  
  
{ Program to demonstrate the InsLine function. }  
  
begin  
    ClrScr;  
    WriteLn;  
    WriteLn(' Line 1' );  
    WriteLn(' Line 2' );  
    WriteLn(' Line 2' );  
    WriteLn(' Line 3' );  
    WriteLn;  
    WriteLn('Oops, Line 2 is listed twice,',  
            ' let''s delete the line at the cursor postion');  
    GotoXY(1,3);  
    ReadKey;  
    DelLine;  
    GotoXY(1,10);  
end.
```

GotoXY

Declaration: `Procedure GotoXY (X: Byte; Y: Byte);`

Description: Positions the cursor at (X,Y), X in horizontal, Y in vertical direction relative to the origin of the current window. The origin is located at (1,1), the upper-left corner of the window.

Errors: None.

See also: [WhereX \(37\)](#), [WhereY \(38\)](#), [Window \(38\)](#)

Listing: crtex/ex6.pp

```
Program Example6;  
uses Crt;  
  
{ Program to demonstrate the GotoXY function. }  
  
begin  
  ClrScr;  
  GotoXY(10,10);  
  Write('10,10');  
  GotoXY(70,20);  
  Write('70,20');  
  GotoXY(1,22);  
end.
```

HighVideo

Declaration: `Procedure HighVideo ;`

Description: HighVideo switches the output to highlighted text. (It sets the high intensity bit of the video attribute)

Errors: None.

See also: [TextColor \(37\)](#), [TextBackground \(36\)](#), [LowVideo \(34\)](#), [NormVideo \(35\)](#)

Listing: crtex/ex14.pp

```
Program Example14;  
uses Crt;  
  
{ Program to demonstrate the LowVideo, HighVideo, NormVideo functions. }  
  
begin  
  LowVideo;  
  WriteLn('This is written with LowVideo');  
  HighVideo;  
  WriteLn('This is written with HighVideo');  
  NormVideo;  
  WriteLn('This is written with NormVideo');  
end.
```

InsLine

Declaration: `Procedure InsLine ;`

Description: InsLine inserts an empty line at the current cursor position. Lines following the current line are scrolled 1 line down, causing the last line to disappear from the window. The cursor doesn't move.

Errors: None.

See also: [ClrEol \(30\)](#), [DelLine \(32\)](#), [ClrScr \(30\)](#)

Listing: crtex/ex10.pp

```
Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
  ClrScr;
  WriteLn;
  WriteLn('Line 1');
  WriteLn('Line 3');
  WriteLn;
  WriteLn('Oops, forgot Line 2, let''s insert at the cursor postion');
  GotoXY(1,3);
  ReadKey;
  InsLine;
  Write('Line 2');
  GotoXY(1,10);
end.
```

KeyPressed

Declaration: `Function KeyPressed : Boolean;`

Description: The Keypressed function scans the keyboard buffer and sees if a key has been pressed. If this is the case, True is returned. If not, False is returned. The Shift, Alt, Ctrl keys are not reported. The key is not removed from the buffer, and can hence still be read after the KeyPressed function has been called.

Errors: None.

See also: [ReadKey \(35\)](#)

Listing: crtex/ex2.pp

```
Program Example2;
uses Crt;

{ Program to demonstrate the KeyPressed function. }

begin
  WriteLn('Waiting until a key is pressed');
  repeat
    until KeyPressed;
  { The key is not Read,
    so it should also be outputted at the commandline }
end.
```

LowVideo

Declaration: `Procedure LowVideo ;`

Description: LowVideo switches the output to non-highlighted text. (It clears the high intensity bit of the video attribute)

Errors: None.

See also: [TextColor \(37\)](#), [TextBackground \(36\)](#), [HighVideo \(33\)](#), [NormVideo \(35\)](#)

For an example, see [HighVideo \(33\)](#)

NormVideo

Declaration: `Procedure NormVideo ;`

Description: NormVideo switches the output to the defaults, read at startup. (The defaults are read from the cursor position at startup)

Errors: None.

See also: [TextColor \(37\)](#), [TextBackground \(36\)](#), [LowVideo \(34\)](#), [HighVideo \(33\)](#)

For an example, see [HighVideo \(33\)](#)

NoSound

Declaration: `Procedure NoSound ;`

Description: Stops the speaker sound. This call is not supported on all operating systems.

Errors: None.

See also: [Sound \(36\)](#)

Listing: `crtex/ex16.pp`

```
Program Example16;
uses Crt;

{ Program to demonstrate the Sound and NoSound function. }

var
  i : longint;
begin
  WriteLn('You will hear some tones from your speaker');
  while (i < 15000) do
    begin
      inc(i, 500);
      Sound(i);
      Delay(100);
    end;
  WriteLn('Quiet now!');
  NoSound; {Stop noise}
end.
```

ReadKey

Declaration: `Function ReadKey : Char;`

Description: The ReadKey function reads 1 key from the keyboard buffer, and returns this. If an extended or function key has been pressed, then the zero ASCII code is returned. You can then read the scan code of the key with a second ReadKey call. **Remark.** Key mappings under Linux can cause the wrong key to be reported by ReadKey, so caution is needed when using ReadKey.

Errors: None.

See also: [KeyPressed \(34\)](#)

Listing: crtex/ex3.pp

```
Program Example3;
uses Crt;

{ Program to demonstrate the ReadKey function. }

var
  ch : char;
begin
  writeln( ' Press Left/Right, Esc=Quit' );
  repeat
    ch:=ReadKey;
    case ch of
      #0 : begin
        ch:=ReadKey; {Read ScanCode}
        case ch of
          #75 : WriteLn( ' Left' );
          #77 : WriteLn( ' Right' );
        end;
      end;
      #27 : WriteLn( 'ESC' );
    end;
  until ch=#27 {Esc}
end.
```

Sound

Declaration: `Procedure Sound (hz : word);`

Description: Sounds the speaker at a frequency of hz. Under WINDOWS, a system sound is played and the frequency parameter is ignored. On other operating systems, this routine may not be implemented.

Errors: None.

See also: [NoSound \(35\)](#)

TextBackground

Declaration: `Procedure TextBackground (CL: Byte);`

Description: TextBackground sets the background color to CL. CL can be one of the predefined color constants.

Errors: None.

See also: [TextColor \(37\)](#), [HighVideo \(33\)](#), [LowVideo \(34\)](#), [NormVideo \(35\)](#)

Listing: crtex/ex13.pp

```
Program Example13;
uses Crt;

{ Program to demonstrate the TextBackground function. }
```

```
begin
  TextColor(White);
  WriteLn('This is written in with the default background color');
  TextBackground(Green);
  WriteLn('This is written in with a Green background');
  TextBackground(Brown);
  WriteLn('This is written in with a Brown background');
  TextBackground(Black);
  WriteLn('Back with a black background');
end.
```

TextColor

Declaration: `Procedure TextColor (CL: Byte);`

Description: TextColor sets the foreground color to CL. CL can be one of the predefined color constants.

Errors: None.

See also: TextBackground (36), HighVideo (33), LowVideo (34), NormVideo (35)

Listing: crtex/ex12.pp

```
Program Example12;
uses Crt;

{ Program to demonstrate the TextColor function. }

begin
  WriteLn('This is written in the default color');
  TextColor(Red);
  WriteLn('This is written in Red');
  TextColor(White);
  WriteLn('This is written in White');
  TextColor(LightBlue);
  WriteLn('This is written in Light Blue');
end.
```

TextMode

Declaration: `procedure TextMode(Mode: Integer);`

Description: TextMode sets the textmode of the screen (i.e. the number of lines and columns of the screen).
The lower byte is use to set the VGA text mode.

This procedure is only implemented on DOS.

Errors: None.

See also: Window (38)

WhereX

Declaration: `Function WhereX : Byte;`

Description: WhereX returns the current X-coordinate of the cursor, relative to the current window. The origin is (1,1), in the upper-left corner of the window.

Errors: None.

See also: [GotoXY \(32\)](#), [WhereY \(38\)](#), [Window \(38\)](#)

Listing: crtex/ex7.pp

```
Program Example7;  
uses Crt;  
  
{ Program to demonstrate the WhereX and WhereY functions. }  
  
begin  
  WriteLn('Cursor position: X=',WhereX,' Y=',WhereY);  
end.
```

WhereY

Declaration: `Function WhereY : Byte;`

Description: WhereY returns the current Y-coordinate of the cursor, relative to the current window. The origin is (1,1), in the upper-left corner of the window.

Errors: None.

See also: [GotoXY \(32\)](#), [WhereX \(37\)](#), [Window \(38\)](#)

Listing: crtex/ex7.pp

```
Program Example7;  
uses Crt;  
  
{ Program to demonstrate the WhereX and WhereY functions. }  
  
begin  
  WriteLn('Cursor position: X=',WhereX,' Y=',WhereY);  
end.
```

Window

Declaration: `Procedure Window (X1, Y1, X2, Y2: Byte);`

Description: Window creates a window on the screen, to which output will be sent. (X1, Y1) are the coordinates of the upper left corner of the window, (X2, Y2) are the coordinates of the bottom right corner of the window. These coordinates are relative to the entire screen, with the top left corner equal to (1,1) Further coordinate operations, except for the next Window call, are relative to the window's top left corner.

Errors: None.

See also: [GotoXY \(32\)](#), [WhereX \(37\)](#), [WhereY \(38\)](#), [ClrScr \(30\)](#)

Listing: crtex/ex5.pp

```
Program Example5;  
uses Crt;  
  
{ Program to demonstrate the Window function. }
```

```
begin
  ClrScr;
  WriteLn('Creating a window from 30,10 to 50,20');
  Window(30,10,50,20);
  WriteLn('We are now writing in this small window we just created, we '+
    'can''t get outside it when writing long lines like this one');
  Write('Press any key to clear the window');
  ReadKey;
  ClrScr;
  Write('The window is cleared, press any key to restore to fullscreen');
  ReadKey;
  {Full Screen is 80x25}
  Window(1,1,80,25);
  Clrscr;
  Writeln('Back in Full Screen');
end.
```

Chapter 2

The DOS unit.

This chapter describes the DOS unit for Free pascal. The DOS unit gives access to some operating system calls related to files, the file system, date and time. Except for the PALMOS target, this unit is available to all supported platforms.

The unit was first written for DOS by Florian klämpfl. It was ported to LINUX by Mark May¹, and enhanced by Michaël Van Canneyt. The AMIGA version was ported by Nils Sjöholm.

Under non-DOS systems, some of the functionality is lost, as it is either impossible or meaningless to implement it. Other than that, the functionality for all operating systems is the same.

This chapter is divided in three sections:

- The first section lists the pre-defined constants, types and variables.
- The second section gives an overview of all functions available, grouped by category.
- The third section describes the functions which appear in the interface part of the DOS unit.

2.1 Types, Variables, Constants

Constants

The DOS unit implements the following constants:

File attributes

The File Attribute constants are used in `FindFirst` (49), `FindNext` (50) to determine what type of special file to search for in addition to normal files. These flags are also used in the `SetFAttr` (58) and `GetFAttr` (53) routines to set and retrieve attributes of files. For their definitions consult table (2.1).

fmXXXX

These constants are used in the Mode field of the `TextRec` record. Gives information on the file-mode of the text I/O. For their definitions consult table (2.2).

¹Current e-mail address mmay@dnaco.net

Table 2.1: Possible file attributes

Constant	Description	Value
readonly	Read only file	\$01
hidden	Hidden file	\$02
sysfile	System file	\$04
volumeid	Volume label	\$08
directory	Directory	\$10
archive	Archive	\$20
anyfile	Any of the above special files	\$3F

Table 2.2: Possible mode constants

Constant	Description	Value
fmclosed	File is closed	\$D7B0
fminput	File is read only	\$D7B1
fmoutput	File is write only	\$D7B2
fminout	File is read and write	\$D7B3

Other

The following constants are not portable, and should not be used. They are present for compatibility only.

```
{Bitmasks for CPU Flags}
fcarry =    $0001;
fparity =   $0004;
fauxiliary = $0010;
fzero =     $0040;
fsign =     $0080;
foverflow  = $0800;
```

Types

The following string types are defined for easy handling of filenames :

```
ComStr  = String[255];    { For command-lines }
PathStr = String[255];    { For full path for file names }
DirStr  = String[255];    { For Directory and (DOS) drive string }
NameStr = String[255];    { For Name of file }
ExtStr  = String[255];    { For Extension of file }
```

```
SearchRec = Packed Record
  Fill : array[1..21] of byte;
  { Fill replaced with declarations below, for Linux}
  Attr : Byte; {attribute of found file}
  Time : LongInt; {last modify date of found file}
  Size : LongInt; {file size of found file}
  Reserved : Word; {future use}
  Name : String[255]; {name of found file}
```

```
SearchSpec: String[255]; {search pattern}
NamePos: Word; {end of path, start of name position}
End;
```

Under LINUX, the Fill array is replaced with the following:

```
SearchNum: LongInt; {to track which search this is}
SearchPos: LongInt; {directory position}
DirPtr: LongInt; {directory pointer for reading directory}
SearchType: Byte; {0=normal, 1=open will close}
SearchAttr: Byte; {attribute we are searching for}
Fill: Array[1..07] of Byte; {future use}
```

This is because the searching mechanism on Unix systems is substantially different from DOS's, and the calls have to be mimicked.

```
const
    filerecnamelength = 255;
type
    FileRec = Packed Record
        Handle,
        Mode,
        RecSize    : longint;
        _private   : array[1..32] of byte;
        UserData   : array[1..16] of byte;
        name       : array[0..filerecnamelength] of char;
    End;
```

FileRec is used for internal representation of typed and untyped files. Text files are handled by the following types :

```
const
    TextRecNameLength = 256;
    TextRecBufSize    = 256;
type
    TextBuf = array[0..TextRecBufSize-1] of char;
    TextRec = Packed Record
        Handle,
        Mode,
        bufsize,
        _private,
        bufpos,
        bufend    : longint;
        bufptr    : ^textbuf;
        openfunc,
        inoutfunc,
        flushfunc,
        closefunc : pointer;
        UserData  : array[1..16] of byte;
        name      : array[0..textrecnamelength-1] of char;
        buffer    : textbuf;
    End;
```

Remark that this is not binary compatible with the Turbo Pascal definition of TextRec, since the sizes of the different fields are different.

```
Registers = record
  case i : integer of
    0 : (ax,f1,bx,f2,cx,f3,dx,f4,bp,f5,si,
         f5i,di,f6,ds,f7,es,f8,flags,fs,gs : word);
    1 : (al,ah,f9,f10,bl,bh,f11,f12,
         cl,ch,f13,f14,dl,dh : byte);
    2 : (eax, ebx, ecx, edx, ebp, esi, edi : longint);
  End;
```

The registers type is used in the MSDos call.

```
DateTime = record
  Year: Word;
  Month: Word;
  Day: Word;
  Hour: Word;
  Min: Word;
  Sec: word;
End;
```

The DateTime type is used in PackTime (57) and UnPackTime (60) for setting/reading file times with GetFTime (53) and SetFTime (58).

Variables

```
DosError : integer;
```

The DosError variable is used by the procedures in the DOS unit to report errors. It can have the following values :

- 2 File not found.
- 3 path not found.
- 5 Access denied.
- 6 Invalid handle.
- 8 Not enough memory.
- 10 Invalid environment.
- 11 Invalid format.
- 18 No more files.

Other values are possible, but are not documented.

2.2 Function list by category

What follows is a listing of the available functions, grouped by category. For each function there is a reference to the page where you can find the function.

File handling

Routines to handle files on disk.

Name	Description	Page
------	-------------	------

FExpand	Expand filename to full path	48
FindClose	Close finfirst/findnext session	49
FindFirst	Start find of file	49
FindNext	Find next file	50
FSearch	Search for file in a path	50
FSplit	Split filename in parts	51
GetFAttr	Return file attributes	53
GetFTime	Return file time	53
GetLongName	Convert short filename to long filename (DOS only)	54
GetShortName	Convert long filename to short filename (DOS only)	55
SetFAttr	Set file attributes	58
SetFTime	Set file time	58

Directory and disk handling

Routines to handle disk information.

Name	Description	Page
AddDisk	Add disk to list of disks (UNIX only)	45
DiskFree	Return size of free disk space	45
DiskSize	Return total disk size	46

Process handling

Functions to handle process information and starting new processes.

Name	Description	Page
DosExitCode	Exit code of last executed program	46
EnvCount	Return number of environment variables	48
EnvStr	Return environment string pair	48
Exec	Execute program	48
GetEnv	Return specified environment string	52

System information

Functions for retrieving and setting general system information such as date and time.

Name	Description	Page
DosVersion	Get OS version	47
GetCBreak	Get setting of control-break handling flag	51
GetDate	Get system date	52
GetIntVec	Get interrupt vector status	54
GetTime	Get system time	55

GetVerify	Get verify flag	56
Intr	Execute an interrupt	56
Keep	Keep process in memory and exit	56
MSDos	Execute MS-dos function call	56
PackTime	Pack time for file time	57
SetCBreak	Set control-break handling flag	57
SetDate	Set system date	58
SetIntVec	Set interrupt vectors	59
SetTime	Set system time	59
SetVerify	Set verify flag	59
SwapVectors	Swap interrupt vectors	59
UnPackTime	Unpack file time	60

2.3 Functions and Procedures

AddDisk

Declaration: `Procedure AddDisk (Const S : String);`

Description: `AddDisk` adds a filename `S` to the internal list of disks. It is implemented for systems which do not use DOS type drive letters. This list is used to determine which disks to use in the `DiskFree` (45) and `DiskSize` (46) calls. The `DiskFree` (45) and `DiskSize` (46) functions need a file on the specified drive, since this is required for the `statfs` system call. The names are added sequentially. The dos initialization code presets the first three disks to:

- `'.'` for the current drive,
- `'/fd0/.'` for the first floppy-drive (linux only).
- `'/fd1/.'` for the second floppy-drive (linux only).
- `'/'` for the first hard disk.

The first call to `AddDisk` will therefore add a name for the second harddisk, The second call for the third drive, and so on until 23 drives have been added (corresponding to drives `'D:'` to `'Z:'`)

Errors: None

See also: `DiskFree` (45), `DiskSize` (46)

DiskFree

Declaration: `Function DiskFree (Drive: byte) : int64;`

Description: `DiskFree` returns the number of free bytes on a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy `a:`, 2 for floppy `b:`, etc. A value of 0 returns the free space on the current drive.

For UNIX only:

The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the dos unit, and have been preset to :

- `'.'` for the current drive,

- `' /fd0 / . '` for the first floppy-drive (linux only).
- `' /fd1 / . '` for the second floppy-drive (linux only).
- `' / '` for the first hard disk.

There is room for 1-26 drives. You can add a drive with the [AddDisk \(45\)](#) procedure. These settings can be coded in `dos.pp`, in the initialization part.

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: [DiskSize \(46\)](#), [AddDisk \(45\)](#)

Listing: `dosex/ex6.pp`

```
Program Example6;  
uses Dos;  
  
{ Program to demonstrate the DiskSize and DiskFree function . }  
  
begin  
  WriteLn('This partition size has ', DiskSize(0), ' bytes');  
  WriteLn('Currently ', DiskFree(0), ' bytes are free');  
end.
```

DiskSize

Declaration: `Function DiskSize (Drive: byte) : int64;`

Description: `DiskSize` returns the total size (in bytes) of a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy a:, 2 for floppy b:, etc. A value of 0 returns the size of the current drive.

For UNIX only:

The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the `dos` unit, and have been preset to :

- `' . '` for the current drive,
- `' /fd0 / . '` for the first floppy-drive (linux only).
- `' /fd1 / . '` for the second floppy-drive (linux only).
- `' / '` for the first hard disk.

There is room for 1-26 drives. You can add a drive with the [AddDisk \(45\)](#) procedure. These settings can be coded in `dos.pp`, in the initialization part.

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: [DiskFree \(45\)](#), [AddDisk \(45\)](#)

For an example, see [DiskFree \(45\)](#).

DosExitCode

Declaration: `Function DosExitCode : Word;`

Description: `DosExitCode` contains (in the low byte) the exit-code of a program executed with the `Exec` call.

Errors: None.

See also: Exec ([48](#))

Listing: dosex/ex5.pp

```
Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

begin
{$IFDEF LINUX}
    WriteLn('Executing /bin/ls -la');
    Exec('/bin/ls',' -la');
{$ELSE}
    WriteLn('Executing Dir');
    Exec(GetEnv('COMSPEC'),' /C dir');
{$ENDIF}
    WriteLn('Program returned with ExitCode ',Lo(DosExitCode));
end.
```

DosVersion

Declaration: Function DosVersion : Word;

Description: DosVersion returns the operating system or kernel version. The low byte contains the major version number, while the high byte contains the minor version number.

Portability: On systems where versions consists of more then two numbers, only the first two numbers will be returned. For example Linux version 2.1.76 will give you DosVersion 2.1. Some operating systems, such as FreeBSD, do not have system calls to return the kernel version, in that case a value of 0 will be returned.

Errors: None.

See also:

Listing: dosex/ex1.pp

```
Program Example1;
uses Dos;

{ Program to demonstrate the DosVersion function. }

var
    OS      : string[32];
    Version : word;
begin
{$IFDEF LINUX}
    OS:='Linux';
{$ENDIF}
{$IFDEF DOS}
    OS:='Dos';
{$ENDIF}
    Version:=DosVersion;
    WriteLn('Current ',OS,' version is ',Lo(Version),'.',Hi(Version));
end.
```

EnvCount

Declaration: `Function EnvCount : longint;`

Description: `EnvCount` returns the number of environment variables.

Errors: None.

See also: `EnvStr` ([48](#)), `Dos:GetEnv` ([52](#))

EnvStr

Declaration: `Function EnvStr (Index: integer) : string;`

Description: `EnvStr` returns the Index-th Name=Value pair from the list of environment variables. The index of the first pair is zero.

Errors: The length is limited to 255 characters.

See also: `EnvCount` ([48](#)), `Dos:GetEnv` ([52](#))

Listing: `dosex/ex13.pp`

Program `Example13;`

uses `Dos;`

{ Program to demonstrate the EnvCount and EnvStr function. }

var

i : `Longint;`

begin

`WriteLn('Current Environment is:');`

for *i*:=1**to** `EnvCount` **do**

`WriteLn(EnvStr(i));`

end.

Exec

Declaration: `Procedure Exec (const Path: pathstr; const ComLine: comstr);`

Description: `Exec` executes the program in `Path`, with the options given by `ComLine`. After the program has terminated, the procedure returns. The Exit value of the program can be consulted with the `DosExitCode` function.

Errors: Errors are reported in `DosError`.

See also: `DosExitCode` ([46](#))

For an example, see `DosExitCode` ([46](#))

FExpand

Declaration: `Function FExpand (const path: pathstr) : pathstr;`

Description: `FExpand` takes its argument and expands it to a complete filename, i.e. a filename starting from the root directory of the current drive, prepended with the drive-letter or volume name (when supported).

Portability: On case sensitive file systems (such as UNIX and LINUX), the resulting name is left as it is, otherwise it is converted to uppercase.

Errors: FSplit ([51](#))

See also:

```
Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

begin
{$IFDEF LINUX}
  WriteLn('Executing /bin/ls -la');
  Exec('/bin/ls',' -la');
{$ELSE}
  WriteLn('Executing Dir');
  Exec(GetEnv('COMSPEC'),' /C dir');
{$ENDIF}
  WriteLn('Program returned with ExitCode ',Lo(DosExitCode));
end.
```

FindClose

Declaration: Procedure FindClose (Var F: SearchRec);

Description: FindClose frees any resources associated with the search record F.

This call is needed to free any internal resources allocated by the FindFirst ([434](#)) or FindNext ([434](#)) calls.

The LINUX implementation of the DOS unit therefore keeps a table of open directories, and when the table is full, closes one of the directories, and reopens another. This system is adequate but slow if you use a lot of searchrecs. So, to speed up the findfirst/findnext system, the FindClose call was implemented. When you don't need a searchrec any more, you can tell this to the DOS unit by issuing a FindClose call. The directory which is kept open for this searchrec is then closed, and the table slot freed.

Portability: It is recommended to use the LINUX call Glob when looking for files on LINUX.

Errors: Errors are reported in DosError.

See also: Glob ([224](#)).

FindFirst

Declaration: Procedure FindFirst (const Path: pathstr; Attr: word; var F: SearchRec);

Description: FindFirst searches the file specified in Path. Normal files, as well as all special files which have the attributes specified in Attr will be returned.

It returns a SearchRec record for further searching in F. Path can contain the wildcard characters ? (matches any single character) and * (matches 0 ore more arbitrary characters). In this case FindFirst will return the first file which matches the specified criteria. If DosError is different from zero, no file(s) matching the criteria was(were) found.

Portability: On OS/2, you cannot issue two different FindFirst calls. That is, you must close any previous search operation with FindClose ([49](#)) before starting a new one. Failure to do so will end in a Run-Time Error 6 (Invalid file handle)

Errors: Errors are reported in `DosError`.

See also: [FindNext \(50\)](#), [FindClose \(49\)](#)

Listing: `dosex/ex7.pp`

```
Program Example7;  
uses Dos;  
  
{ Program to demonstrate the FindFirst and FindNext function. }  
  
var  
  Dir : SearchRec;  
begin  
  FindFirst( '*..*', archive, Dir );  
  WriteLn( ' FileName'+Space(32), ' FileSize':9 );  
  while ( DosError=0 ) do  
    begin  
      WriteLn( Dir.Name+Space(40-Length( Dir.Name )), Dir.Size:9 );  
      FindNext( Dir );  
    end;  
  FindClose( Dir );  
end.
```

FindNext

Declaration: `Procedure FindNext (var f: searchRec);`

Description: `FindNext` takes as an argument a `SearchRec` from a previous `FindNext` call, or a `FindFirst` call, and tries to find another file which matches the criteria, specified in the `FindFirst` call. If `DosError` is different from zero, no more files matching the criteria were found.

Errors: `DosError` is used to report errors.

See also: [FindFirst \(49\)](#), [FindClose \(49\)](#)

For an example, see [FindFirst \(49\)](#).

FSearch

Declaration: `Function FSearch (Path: pathstr; DirList: string) : pathstr;`

Description: `FSearch` searches the file `Path` in all directories listed in `DirList`. The full name of the found file is returned. `DirList` must be a list of directories, separated by semi-colons. When no file is found, an empty string is returned.

Portability: On UNIX systems, `DirList` can also be separated by colons, as is customary on those environments.

Errors: None.

See also: [FExpand \(48\)](#)

Listing: `dosex/ex10.pp`

```
Program Example10;  
uses Dos;  
  
{ Program to demonstrate the FSearch function. }  
  
var  
  s : string;  
begin  
  s:=FSearch(ParamStr(1),GetEnv('PATH'));  
  if s='' then  
    WriteLn(ParamStr(1),' not Found in PATH')  
  else  
    WriteLn(ParamStr(1),' Found in PATH at ',s);  
end.
```

FSplit

Declaration: `Procedure FSplit (path: pathstr;
var dir: dirstr; var name: namestr; var ext: extstr);`

Description: FSplit splits a full file name into 3 parts : A Path, a Name and an extension (in ext.) The extension is taken to be all letters after the *last* dot (.). For DOS, however, an exception is made when LFNSupport=False, then the extension is defined as all characters after the *first* dot.

Errors: None.

See also: FSearch ([50](#))

Listing: dosex/ex12.pp

```
Program Example12;  
uses Dos;  
  
{ Program to demonstrate the FSplit function. }  
  
var  
  Path,Name,Ext : string;  
begin  
  FSplit(ParamStr(1),Path,Name,Ext);  
  WriteLn(' Splitted ',ParamStr(1),' in :');  
  WriteLn(' Path      : ',Path);  
  WriteLn(' Name      : ',Name);  
  WriteLn(' Extension : ',Ext);  
end.
```

GetCBreak

Declaration: `Procedure GetCBreak (var breakvalue: boolean);`

Description: GetCBreak gets the status of CTRL-Break checking under DOS and AMIGA. When BreakValue is false, then DOS only checks for the CTRL-Break key-press when I/O is performed. When it is set to True, then a check is done at every system call.

Portability: Under non-DOS and non-AMIGA operating systems, BreakValue always returns True.

Errors: None

See also: SetCBreak ([57](#))

GetDate

Declaration: `Procedure GetDate (var year, month, mday, wday: word);`

Description: `GetDate` returns the system's date. Year is a number in the range 1980..2099. mday is the day of the month, wday is the day of the week, starting with Sunday as day 0.

Errors: None.

See also: `GetTime` ([55](#)), `SetDate` ([58](#))

Listing: `dosex/ex2.pp`

```
Program Example2;
uses Dos;

{ Program to demonstrate the GetDate function. }

const
  DayStr: array [0..6] of string [3] = ( 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat' );
  MonthStr: array [1..12] of string [3] = ( 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                                             'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' );

var
  Year, Month, Day, WDay : word;
begin
  GetDate (Year, Month, Day, WDay);
  WriteLn ( ' Current date ' );
  WriteLn ( DayStr[WDay], ' ', Day, ' ', MonthStr[Month], ' ', Year, ' ' );
end.
```

GetEnv

Declaration: `Function GetEnv (EnvVar: String) : String;`

Description: `Getenv` returns the value of the environment variable `EnvVar`. When there is no environment variable `EnvVar` defined, an empty string is returned.

Portability: Under some operating systems (such as UNIX), case is important when looking for `EnvVar`.

Errors: None.

See also: `EnvCount` ([48](#)), `EnvStr` ([48](#))

Listing: `dosex/ex14.pp`

```
Program Example14;
uses Dos;

{ Program to demonstrate the GetEnv function. }

begin
  WriteLn ( ' Current PATH is ', GetEnv ( 'PATH' ) );
end.
```

GetFAttr

Declaration: `Procedure GetFAttr (var F; var Attr: word);`

Description: `GetFAttr` returns the file attributes of the file-variable `f`. `F` can be a untyped or typed file, or of type `Text`. `f` must have been assigned, but not opened. The attributes can be examined with the following constants :

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Under LINUX, supported attributes are:

- `Directory`
- `ReadOnly` if the current process doesn't have access to the file.
- `Hidden` for files whose name starts with a dot (`'.'`).

Errors: Errors are reported in `DosError`

See also: `SetFAttr` ([58](#))

Listing: `dosex/ex8.pp`

```
Program Example8;
uses Dos;

{ Program to demonstrate the GetFAttr function. }

var
  Attr : Word;
  f    : File;
begin
  Assign(f, ParamStr(1));
  GetFAttr(f, Attr);
  WriteLn('File ', ParamStr(1), ' has attribute ', Attr);
  if (Attr and archive) <> 0 then WriteLn('– Archive');
  if (Attr and directory) <> 0 then WriteLn('– Directory');
  if (Attr and readonly) <> 0 then WriteLn('– Read-Only');
  if (Attr and sysfile) <> 0 then WriteLn('– System');
  if (Attr and hidden) <> 0 then WriteLn('– Hidden');
end.
```

GetFTime

Declaration: `Procedure GetFTime (var F; var Time: longint);`

Description: `GetFTime` returns the modification time of a file. This time is encoded and must be decoded with `UnPackTime`. `F` must be a file type, which has been assigned, and opened.

Errors: Errors are reported in `DosError`

See also: `SetFTime` ([58](#)), `PackTime` ([57](#)), `UnPackTime` ([60](#))

Listing: dosex/ex9.pp

```
Program Example9;
uses Dos;

{ Program to demonstrate the GetFTime function. }

Function L0(w: word): string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
    L0:= '0'+s
  else
    L0:=s;
end;

var
  f      : File;
  Time   : Longint;
  DT     : DateTime;
begin
  Assign(f, ParamStr(1));
  Reset(f);
  GetFTime(f, Time);
  Close(f);
  UnPackTime(Time,DT);
  Write('File ', ParamStr(1), ' is last modified on ');
  Writeln(L0(DT.Month), '-', L0(DT.Day), '-', DT.Year,
          ' at ', L0(DT.Hour), ':', L0(DT.Min));
end.
```

GetIntVec

Declaration: Procedure GetIntVec (IntNo: byte; var Vector: pointer);

Description: GetIntVec returns the address of interrupt vector IntNo.

Portability: This call does nothing, it is present for compatibility only.

Errors: None.

See also: SetIntVec ([59](#))

GetLongName

Declaration: function GetLongName(var p : String) : boolean;

Description: This function is only implemented in the GO32V2 version of Free Pascal.

GetLongName changes the filename p to a long filename if the DOS call to do this is successful. The resulting string is the long file name corresponding to the short filename p.

The function returns True if the DOS call was successful, False otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the DOS call was not succesfull, False is returned.

See also: GetShortName ([55](#))

GetShortName

Declaration: `function GetShortName(var p : String) : boolean;`

Description: This function is only implemented in the GO32V2 version of Free Pascal.

`GetShortName` changes the filename `p` to a short filename if the DOS call to do this is successful. The resulting string is the short file name corresponding to the long filename `p`.

The function returns `True` if the DOS call was successful, `False` otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the DOS call was not successful, `False` is returned.

See also: `GetLongName` ([54](#))

GetTime

Declaration: `Procedure GetTime (var hour, minute, second, sec100: word);`

Description: `GetTime` returns the system's time. `Hour` is on a 24-hour time scale. `sec100` is in hundredth of a second.

Portability: Certain operating systems (such as AMIGA), always set the `sec100` field to zero.

Errors: None.

See also: `GetDate` ([52](#)), `SetTime` ([59](#))

Listing: `dosex/ex3.pp`

```
Program Example3;
uses Dos;

{ Program to demonstrate the GetTime function. }

Function L0(w:word):string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
    L0:='0'+s
  else
    L0:=s;
end;

var
  Hour,Min,Sec,HSec : word;
begin
  GetTime(Hour,Min,Sec,HSec);
  WriteLn(' Current time ');
  WriteLn(L0(Hour),':',L0(Min),':',L0(Sec));
end.
```

GetVerify

Declaration: `Procedure GetVerify (var verify: boolean);`

Description: `GetVerify` returns the status of the verify flag under DOS. When `Verify` is `True`, then DOS checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Portability: Under non-DOS systems (excluding OS/2 applications running under vanilla DOS), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([59](#))

Intr

Declaration: `Procedure Intr (IntNo: byte; var Regs: registers);`

Description: `Intr` executes a software interrupt number `IntNo` (must be between 0 and 255), with processor registers set to `Regs`. After the interrupt call returned, the processor registers are saved in `Regs`.

Portability: Under non-DOS operating systems, this call does nothing.

Errors: None.

See also: `MSDos` ([56](#)), see the `LINUX` unit.

Keep

Declaration: `Procedure Keep (ExitCode: word);`

Description: `Keep` terminates the program, but stays in memory. This is used for TSR (Terminate Stay Resident) programs which catch some interrupt. `ExitCode` is the same parameter as the `Halt` function takes.

Portability: This call does nothing, it is present for compatibility only.

Errors: None.

See also: `Halt` ()

MSDos

Declaration: `Procedure MSdos (var regs: registers);`

Description: `MSdos` executes an operating system. This is the same as doing a `Intr` call with the interrupt number for an os call.

Portability: Under non-DOS operating systems, this call does nothing. On DOS systems, this calls interrupt \$21.

Errors: None.

See also: `Intr` ([56](#))

PackTime

Declaration: `Procedure PackTime (var T: datetime; var P: longint);`

Description: `UnPackTime` converts the date and time specified in `T` to a packed-time format which can be fed to `SetFTime`.

Errors: None.

See also: `SetFTime` ([58](#)), `FindFirst` ([49](#)), `FindNext` ([50](#)), `UnPackTime` ([60](#))

Listing: `dosex/ex4.pp`

```
Program Example4;
uses Dos;

{ Program to demonstrate the PackTime and UnPackTime functions. }

var
    DT    : DateTime;
    Time  : longint;
begin
    with DT do
        begin
            Year:=1998;
            Month:=11;
            Day:=11;
            Hour:=11;
            Min:=11;
            Sec:=11;
        end;
    PackTime(DT,Time);
    WriteLn('Packed Time : ',Time);
    UnPackTime(Time,DT);
    WriteLn('Unpacked Again: ');
    with DT do
        begin
            WriteLn('Year   ',Year);
            WriteLn('Month  ',Month);
            WriteLn('Day    ',Day);
            WriteLn('Hour   ',Hour);
            WriteLn('Min    ',Min);
            WriteLn('Sec    ',Sec);
        end;
end.
```

SetCBreak

Declaration: `Procedure SetCBreak (breakvalue: boolean);`

Description: `SetCBreak` sets the status of CTRL-Break checking. When `BreakValue` is `false`, then DOS only checks for the CTRL-Break key-press when I/O is performed. When it is set to `True`, then a check is done at every system call.

Portability: Under non-DOS and non-AMIGA operating systems, this call does nothing.

Errors: None.

See also: `GetCBreak` ([51](#))

SetDate

Declaration: `Procedure SetDate (year, month, day: word);`

Description: `SetDate` sets the system's internal date. Year is a number between 1980 and 2099.

Portability: On a LINUX machine, there must be root privileges, otherwise this routine will do nothing. On other UNIX systems, this call currently does nothing.

Errors: None.

See also: `Dos:GetDate` ([52](#)), `SetTime` ([59](#))

SetFAttr

Declaration: `Procedure SetFAttr (var F; Attr: word);`

Description: `SetFAttr` sets the file attributes of the file-variable F. F can be a untyped or typed file, or of type `Text`. F must have been assigned, but not opened. The attributes can be a sum of the following constants:

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Portability: Under UNIX like systems (such as LINUX and BEOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`.

See also: `GetFAttr` ([53](#))

SetFTime

Declaration: `Procedure SetFTime (var F; Time: longint);`

Description: `SetFTime` sets the modification time of a file, this time is encoded and must be encoded with `PackTime`. F must be a file type, which has been assigned, and opened.

Portability: Under UNIX like systems (such as LINUX and BEOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`

See also: `GetFTime` ([53](#)), `PackTime` ([57](#)), `UnPackTime` ([60](#))

SetIntVec

Declaration: `Procedure SetIntVec (IntNo: byte; Vector: pointer);`

Description: `SetIntVec` sets interrupt vector `IntNo` to `Vector`. `Vector` should point to an interrupt procedure.

Portability: This call does nothing, it is present for compatibility only.

Errors: None.

See also: `GetIntVec` ([54](#))

SetTime

Declaration: `Procedure SetTime (hour,minute,second,sec100: word);`

Description: `SetTime` sets the system's internal clock. The `Hour` parameter is on a 24-hour time scale.

Portability: On a LINUX machine, there must be root privileges, otherwise this routine will do nothing. On other UNIX systems, this call currently does nothing.

Errors: None.

See also: `Dos:GetTime` ([55](#)), `SetDate` ([58](#))

SetVerify

Declaration: `Procedure SetVerify (verify: boolean);`

Description: `SetVerify` sets the status of the verify flag under DOS. When `Verify` is `True`, then DOS checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Portability: Under non-DOS operating systems (excluding OS/2 applications running under vanilla dos), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([59](#))

SwapVectors

Declaration: `Procedure SwapVectors ;`

Description: `SwapVectors` swaps the contents of the internal table of interrupt vectors with the current contents of the interrupt vectors. This is called typically in before and after an `Exec` call.

Portability: Under certain operating systems, this routine may be implemented as an empty stub.

Errors: None.

See also: `Exec` ([48](#)), `SetIntVec` ([59](#))

UnPackTime

Declaration: `Procedure UnPackTime (p: longint; var T: datetime);`

Description: `UnPackTime` converts the file-modification time in `p` to a `DateTime` record. The file-modification time can be returned by `GetFTime`, `FindFirst` or `FindNext` calls.

Errors: None.

See also: `GetFTime` ([53](#)), `FindFirst` ([49](#)), `FindNext` ([50](#)), `PackTime` ([57](#))

For an example, see `PackTime` ([57](#)).

Chapter 3

The DXELOAD unit

3.1 Introduction

The `dxeload` unit was implemented by Pierre Müller for DOS, it allows to load a DXE file (an object file with 1 entry point) into memory and return a pointer to the entry point.

It exists only for DOS.

3.2 Constants, types and variables

Constants

The following constant is the magic number, found in the header of a DXE file.

```
DXE_MAGIC = $31455844;
```

Types

The following record describes the header of a DXE file. It is used to determine the magic number of the DXE file and number of relocations that must be done when the object file is loaded in memory.

```
dx_header = record
    magic,
    symbol_offset,
    element_size,
    nrelocs      : longint;
end;
```

3.3 Functions and Procedures

`dx_load`

Declaration: `function dx_load(filename : string) : pointer;`

Description: `dx_load` loads the contents of the file `filename` into memory. It performs the necessary relocations in the object code, and returns then a pointer to the entry point of the code.

Errors: If an error occurs during the load or relocations, `Nil` is returned.

For an example, see the `emu387` unit in the RTL.

Chapter 4

The EMU387 unit

The `emu387` unit was written by Pierre Müller for DOS. It sets up the coprocessor emulation for FPC under DOS. It is not necessary to use this unit on other OS platforms because they either simply do not run on a machine without coprocessor, or they provide the coprocessor emulation themselves.

It shouldn't be necessary to use the function in this unit, it should be enough to place this unit in the `uses` clause of your program to enable the coprocessor emulation under DOS. The unit initialization code will try and load the coprocessor emulation code and initialize it.

4.1 Functions and procedures

npxsetup

Declaration: `procedure npxsetup(prog_name : string);`

Description: `npxsetup` checks whether a coprocessor is found. If not, it loads the file `wmemu387.dxe` into memory and initializes the code in it.

If the environment variable `387` is set to `N`, then the emulation will be loaded, even if there is a coprocessor present. If the variable doesn't exist, or is set to any other value, the unit will try to detect the presence of a coprocessor unit.

The function searches the file `wmemu387.dxe` in the following way:

- 1.If the environment variable `EMU387` is set, then it is assumed to point at the `wmemu387.dxe` file.
- 2.if the environment variable `EMU387` does not exist, then the function will take the path part of `prog_name` and look in that directory for the file `wmemu387.dxe`.

It should never be necessary to call this function, because the initialization code of the unit contains a call to the function with as an argument `paramstr(0)`. This means that you should deliver the file `wmemu387.dxe` together with your program.

Errors: If there is an error, an error message is printed to standard error, and the program is halted, since any floating-point code is bound to fail anyhow.

Chapter 5

The GETOPTS unit.

This document describes the GETOPTS unit for Free Pascal. It was written for LINUX by Michaël Van Canneyt. It now also works for all supported platforms.

The getopt unit provides a mechanism to handle command-line options in a structured way, much like the GNU getopt mechanism. It allows you to define the valid options for your program, and the unit will then parse the command-line options for you, and inform you of any errors.

The chapter is divided in 2 sections:

- The first section lists types, constants and variables from the interface part of the unit.
- The second section describes the functions defined in the unit.

5.1 Types, Constants and variables :

Constants

No_Argument=0 : Specifies that a long option does not take an argument.

Required_Argument=1 : Specifies that a long option needs an argument.

Optional_Argument=2 : Specifies that a long option optionally takes an argument.

EndOfOptions=#255 : Returned by getopt, getlongopts to indicate that there are no more options.

Types

```
TOption = record
  Name      : String;
  Has_arg   : Integer;
  Flag      : PChar;
  Value     : Char;
end;
POption = ^TOption;
```

The option type is used to communicate the long options to GetLongOpts. The Name field is the name of the option. Has_arg specifies if the option wants an argument, Flag is a pointer to a char, which is set to Value, if it is non-nil. POption is a pointer to a Option record. It is used as an argument to the GetLongOpts function.

Variables

`OptArg: String` Is set to the argument of an option, if the option needs one.

`Optind: Longint` Is the index of the current `paramstr()`. When all options have been processed, `optind` is the index of the first non-option parameter. This is a read-only variable. Note that it can become equal to `paramcount+1`

`OptErr: Boolean` Indicates whether `getopt()` prints error messages.

`OptOpt: Char` In case of an error, contains the character causing the error.

5.2 Procedures and functions

GetLongOpts

Declaration: `Function GetLongOpts (Shortopts : String, LongOpts : POption; var Longint : Longint) : Char;`

Description: Returns the next option found on the command-line, taking into account long options as well. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable. `ShortOptions` is a string containing all possible one-letter options. (see [Getopt \(65\)](#) for its description and use) `LongOpts` is a pointer to the first element of an array of `Option` records, the last of which needs a name of zero length. The function tries to match the names even partially (i.e. `-app` will match e.g. the append option), but will report an error in case of ambiguity. If the option needs an argument, set `Has_arg` to `Required_argument`, if the option optionally has an argument, set `Has_arg` to `Optional_argument`. If the option needs no argument, set `Has_arg` to zero. Required arguments can be specified in two ways :

1. Pasted to the option : `-option=value`
2. As a separate argument : `-option value`

Optional arguments can only be specified through the first method.

Errors: see [Getopt \(65\)](#), `getopt (3)`

See also: `Getopt`

Getopt

Declaration: `Function Getopt (Shortopts : String) : Char;`

Description: Returns the next option found on the command-line. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable. `ShortOptions` is a string containing all possible one-letter options. If a letter is followed by a colon (:), then that option needs an argument. If a letter is followed by 2 colons, the option has an optional argument. If the first character of `shortoptions` is a '+' then options following a non-option are regarded as non-options (standard Unix behavior). If it is a '-', then all non-options are treated as arguments of a option with character #0. This is useful for applications that require their options in the exact order as they appear on the command-line. If the first character of `shortoptions` is none of the above, options and non-options are permuted, so all non-options are behind all options. This allows options and non-options to be in random order on the command line.

Errors: Errors are reported through giving back a '?' character. `OptOpt` then gives the character which caused the error. If `OptErr` is `True` then `getopt` prints an error-message to `stdout`.

See also: [GetLongOpts \(65\)](#), `getopt (3)`

```
program testopt;  
  
  { Program to depmonstrate the getopt function. }  
  
  {  
    Valid calls to this program are  
    optex --verbose --add me --delete you  
    optex --append --create child  
    optex -ab -c me -d you  
    and so on  
  }  
uses getopt;  
  
var c : char;  
    optionindex : Longint;  
    theopts : array[1..7] of TOption;  
  
begin  
  with theopts[1] do  
    begin  
      name := 'add';  
      has_arg := 1;  
      flag := nil;  
      value := #0;  
    end;  
  with theopts[2] do  
    begin  
      name := 'append';  
      has_arg := 0;  
      flag := nil;  
      value := #0;  
    end;  
  with theopts[3] do  
    begin  
      name := 'delete';  
      has_arg := 1;  
      flag := nil;  
      value := #0;  
    end;  
  with theopts[4] do  
    begin  
      name := 'verbose';  
      has_arg := 0;  
      flag := nil;  
      value := #0;  
    end;  
  with theopts[5] do  
    begin  
      name := 'create';  
      has_arg := 1;  
      flag := nil;  
      value := 'c'  
    end;  
  with theopts[6] do  
    begin  
      name := 'file';  
      has_arg := 1;  
      flag := nil;
```

```
    value:=#0;
end;
with theopts[7] do
  begin
    name:='';
    has_arg:=0;
    flag:=nil;
  end;
c:=#0;
repeat
  c:=getlongopts('abc:d:012',@theo[1],optionindex);
  case c of
    '1','2','3','4','5','6','7','8','9' :
      begin
        writeln('Got optind : ',c)
      end;
    #0 : begin
        write('Long option : ',theo[optionindex].name);
        if theopts[optionindex].has_arg>0 then
          writeln(' With value : ',optarg)
        else
          writeln
        end;
    'a' : writeln('Option a. ');
    'b' : writeln('Option b. ');
    'c' : writeln('Option c : ',optarg);
    'd' : writeln('Option d : ',optarg);
    '?',':' : writeln('Error with opt : ',optopt);
  end; { case }
until c=endofoptions;
if optind<=paramcount then
  begin
    write('Non options : ');
    while optind<=paramcount do
      begin
        write(paramstr(optind), ' ');
        inc(optind)
      end;
    writeln
  end
end.
```

Chapter 6

The GPM unit

6.1 Introduction

The GPM unit implements an interface to filelibgpm, the console program for mouse handling. This unit was created by Peter Vreman, and is only available on LINUX.

When this unit is used, your program is linked to the C libraries, so you must take care of the C library version. Also, it will only work with version 1.17 or higher of the libgpm library.

6.2 Constants, types and variables

constants

The following constants are used to denote filenames used by the library:

```
_PATH_VARRUN = '/var/run/';  
_PATH_DEV    = '/dev/';  
GPM_NODE_DIR = _PATH_VARRUN;  
GPM_NODE_DIR_MODE = 0775;  
GPM_NODE_PID  = '/var/run/gpm.pid';  
GPM_NODE_DEV  = '/dev/gpmctl';  
GPM_NODE_CTL  = GPM_NODE_DEV;  
GPM_NODE_FIFO = '/dev/gpmdata';
```

The following constants denote the buttons on the mouse:

```
GPM_B_LEFT    = 4;  
GPM_B_MIDDLE  = 2;  
GPM_B_RIGHT   = 1;
```

The following constants define events:

```
GPM_MOVE = 1;  
GPM_DRAG = 2;  
GPM_DOWN = 4;  
GPM_UP   = 8;  
GPM_SINGLE = 16;  
GPM_DOUBLE = 32;
```

```
GPM_TRIPLE = 64;  
GPM_MFLAG = 128;  
GPM_HARD = 256;  
GPM_ENTER = 512;  
GPM_LEAVE = 1024;
```

The following constants are used in defining margins:

```
GPM_TOP = 1;  
GPM_BOT = 2;  
GPM_LFT = 4;  
GPM_RGT = 8;
```

Types

The following general types are defined:

```
TGpmEtype = longint;  
TGpmMargin = longint;
```

The following type describes an event; it is passed in many of the gpm functions.

```
PGpmEvent = ^TGpmEvent;  
TGpmEvent = record  
  buttons : byte;  
  modifiers : byte;  
  vc : word;  
  dx : word;  
  dy : word;  
  x : word;  
  y : word;  
  EventType : TGpmEType;  
  clicks : longint;  
  margin : TGpmMargin;  
end;  
TGpmHandler=function(var event:TGpmEvent;clientdata:pointer):longint;cdecl;
```

The following types are used in connecting to the gpm server:

```
PGpmConnect = ^TGpmConnect;  
TGpmConnect = record  
  eventMask : word;  
  defaultMask : word;  
  minMod : word;  
  maxMod : word;  
  pid : longint;  
  vc : longint;  
end;
```

The following type is used to define *regions of interest*

```
PGpmRoi = ^TGpmRoi;  
TGpmRoi = record
```

```
xMin : integer;  
xMax : integer;  
yMin : integer;  
yMax : integer;  
minMod : word;  
maxMod : word;  
eventMask : word;  
owned : word;  
handler : TGpmHandler;  
clientdata : pointer;  
prev : PGpmRoi;  
next : PGpmRoi;  
end;
```

Variables

The following variables are imported from the gpm library

```
gpm_flag          : longint;cvar;external;  
gpm_fd            : longint;cvar;external;  
gpm_hflag         : longint;cvar;external;  
gpm_morekeys     : Longbool;cvar;external;  
gpm_zerobased    : Longbool;cvar;external;  
gpm_visiblepointer : Longbool;cvar;external;  
gpm_mx           : longint;cvar;external;  
gpm_my           : longint;cvar;external;  
gpm_timeout      : TTimeVal;cvar;external;  
_gpm_buf         : array[0..0] of char;cvar;external;  
_gpm_arg         : ^word;cvar;external;  
gpm_handler      : TGpmHandler;cvar;external;  
gpm_data         : pointer;cvar;external;  
gpm_roi_handler  : TGpmHandler;cvar;external;  
gpm_roi_data     : pointer;cvar;external;  
gpm_roi          : PGpmRoi;cvar;external;  
gpm_current_roi  : PGpmRoi;cvar;external;  
gpm_consolefd    : longint;cvar;external;  
Gpm_HandleRoi    : TGpmHandler;cvar;external;
```

6.3 Functions and procedures

Gpm_AnyDouble

Declaration: `function Gpm_AnyDouble(EventType : longint) : boolean;`

Description: `Gpm_AnyDouble` returns True if `EventType` contains the `GPM_DOUBLE` flag, False otherwise.

Errors: None.

See also: `Gpm_StrictSingle` ([75](#)), `Gpm_AnySingle` ([71](#)), `Gpm_StrictDouble` ([75](#)), `Gpm_StrictTriple` ([75](#)), `Gpm_AnyTriple` ([71](#))

Gpm_AnySingle

Declaration: `function Gpm_AnySingle(EventType : longint) : boolean;`

Description: `Gpm_AnySingle` returns True if `EventType` contains the `GPM_SINGLE` flag, False otherwise.

Errors:

See also: `Gpm_StrictSingle` (75), `Gpm_AnyDoubmle` (70), `Gpm_StrictDouble` (75), `Gpm_StrictTriple` (75), `Gpm_AnyTriple` (71)

Gpm_AnyTriple

Declaration: `function Gpm_AnyTriple(EventType : longint) : boolean;`

Description:

Errors:

See also: `Gpm_StrictSingle` (75), `Gpm_AnyDoubmle` (70), `Gpm_StrictDouble` (75), `Gpm_StrictTriple` (75), `Gpm_AnySingle` (71)

Gpm_Close

Declaration: `function Gpm_Close:longint;cdecl;external;`

Description: `Gpm_Close` closes the current connection, and pops the connection stack; this means that the previous connection becomes active again.

The function returns -1 if the current connection is not the last one, and it returns 0 if the current connection is the last one.

Errors: None.

See also: `Gpm_Open` (74)

for an example, see `Gpm_GetEvent` (72).

Gpm_FitValues

Declaration: `function Gpm_FitValues(var x,y:longint):longint;cdecl;external;`

Description: `Gpm_fitValues` changes `x` and `y` so they fit in the visible screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: `Gpm_FitValuesM` (71),

Gpm_FitValuesM

Declaration: `function Gpm_FitValuesM(var x,y:longint; margin:longint):longint;cdecl;external;`

Description: `Gpm_FitValuesM` chnages `x` and `y` so they fit in the margin indicated by `margin`. If `margin` is -1, then the values are fitted to the screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: `Gpm_FitValues` (71),

Gpm_GetEvent

Declaration: `function Gpm_GetEvent(var Event:TGpmEvent):longint;cdecl;external;`

Description: `Gpm_GetEvent` Reads an event from the file descriptor `gpm_fd`. This file is only for internal use and should never be called by a client application.

It returns 1 on succes, and -1 on failue.

Errors: On error, -1 is returned.

See also: `seeflGpm_GetSnapshotGpmGetSnapshot`

Listing: gpmex/gpmex.pp

```
program gpmex;

{
  Example program to demonstrate the use of the gpm unit.
}

uses gpm;

var
  connect : TGPMConnect;
  event   : tgpmevent;

begin
  connect.EventMask:=GPM_MOVE or GPM_DRAG or GPM_DOWN or GPM_UP;
  connect.DefaultMask:=0;
  connect.MinMod:=0;
  connect.MaxMod:=0;
  if Gpm_Open(connect,0)=-1 then
    begin
      WriteLn('No mouse handler present. ');
      Halt(1);
    end;
  WriteLn('Click right button to end. ');
  Repeat
    gpm_getevent(Event);
    With Event do
      begin
        Write('Pos = (',X,',',Y,',') Buttons : ( ');
        if (buttons and Gpm_b_left)<>0 then
          write(' left ');
        if (buttons and Gpm_b_right)<>0 then
          write(' right ');
        if (buttons and Gpm_b_middle)<>0 then
          Write(' middle ');
        Write(') Event : ');
        Case EventType and $F of
          GPM_MOVE: write(' Move ');
          GPM_DRAG: write(' Drag ');
          GPM_DOWN: write(' Down ');
          GPM_UP: write(' Up ');
        end;
        WriteLn;
      end;
    Until (Event.Buttons and gpm_b_right)<>0;
    gpm_close;
  end.
```

Gpm_GetLibVersion

Declaration: `function Gpm_GetLibVersion(var where:longint):pchar;cdecl;external;`

Description: `Gpm_GetLibVersion` returns a pointer to a version string, and returns in `where` an integer representing the version. The version string represents the version of the gpm library.

The return value is a `pchar`, which should not be deallocated, i.e. it is not on the heap.

Errors: None.

See also: `Gpm_GetServerVersion` (73)

Gpm_GetServerVersion

Declaration: `function Gpm_GetServerVersion(var where:longint):pchar;cdecl;external;`

Description: `Gpm_GetServerVersion` returns a pointer to a version string, and returns in `where` an integer representing the version. The version string represents the version of the gpm server program.

The return value is a `pchar`, which should not be deallocated, i.e. it is not on the heap.

Errors: If the gpm program is not present, then the function returns `Nil`

See also: `Gpm_GetLibVersion` (73)

Gpm_GetSnapshot

Declaration: `function Gpm_GetSnapshot(var Event:TGpmEvent):longint;cdecl;external;`

Description: `Gpm_GetSnapshot` returns the picture that the server has of the current situation in `Event`. This call will not read the current situation from the mouse file descriptor, but returns a buffered version.

The meaning of the fields is as follows:

x,ycurrent position of the cursor.

dx,dysize of the window.

vcnumber of the virtual console.

modifierskeyboard shift state.

buttonsbuttons which are currently pressed.

clicksnumber of clicks (0,1 or 2).

The function returns the number of mouse buttons, or -1 if this information is not available.

Errors: None.

See also: `Gpm_GetEvent` (72)

Gpm_LowerRoi

Declaration: `function Gpm_LowerRoi(which:PGpmRoi; after:PGpmRoi):PGpmRoi;cdecl;external;`

Description: `Gpm_LowerRoi` lowers the region of interest `which` after `after`. If `after` is `Nil`, the region of interest is moved to the bottom of the stack.

The return value is the new top of the region-of-interest stack.

Errors: None.

See also: `Gpm_RaiseRoi` (74), `Gpm_PopRoi` (74), `Gpm_PushRoi` (74)

Gpm_Open

Declaration: `function Gpm_Open(var Conn:TGpmConnect; Flag:longint):longint;cdecl;external;`

Description: `Gpm_Open` opens a new connection to the mouse server. The connection is described by the fields of the `conn` record:

EventMaskA bitmask of the events the program wants to receive.

DefaultMaskA bitmask to tell the library which events get their default treatment (text selection).

minModthe minimum amount of modifiers needed by the program.

maxModthe maximum amount of modifiers needed by the program.

if `Flag` is 0, then the application only receives events that come from its own terminal device. If it is negative it will receive all events. If the value is positive then it is considered a console number to which to connect.

The return value is -1 on error, or the file descriptor used to communicate with the client. Under an X-Term the return value is -2.

Errors: On Error, the return value is -1.

See also: `Gpm_Open` (74)

for an example, see `Gpm_GetEvent` (72).

Gpm_PopRoi

Declaration: `function Gpm_PopRoi(which:PGpmRoi):PGpmRoi;cdecl;external;`

Description: `Gpm_PopRoi` pops the topmost region of interest from the stack. It returns the next element on the stack, or `Nil` if the current element was the last one.

Errors: None.

See also: `Gpm_RaiseRoi` (74), `Gpm_LowerRoi` (73), `Gpm_PushRoi` (74)

Gpm_PushRoi

Declaration: `function Gpm_PushRoi(x1:longint; y1:longint; X2:longint; Y2:longint; mask:longint; fun:TGpmHandler; xtradata:pointer):PGpmRoi;cdecl;external;`

Description: `Gpm_PushRoi` puts a new *region of interest* on the stack. The region of interest is defined by a rectangle described by the corners `(X1,Y1)` and `(X2,Y2)`.

The mask describes which events the handler `fun` will handle; `ExtraData` will be put in the `xtradata` field of the `TGPM_Roi` record passed to the fun handler.

Errors: None.

See also: `Gpm_RaiseRoi` (74), `Gpm_PopRoi` (74), `Gpm_LowerRoi` (73)

Gpm_RaiseRoi

Declaration: `function Gpm_RaiseRoi(which:PGpmRoi; before:PGpmRoi):PGpmRoi;cdecl;external;`

Description: `Gpm_RaiseRoi` raises the *region of interest* which till it is on top of region `before`. If `before` is `nil` then the region is put on top of the stack. The returned value is the top of the stack.

Errors: None.

See also: `Gpm_PushRoi` (74), `Gpm_PopRoi` (74), `Gpm_LowerRoi` (73)

Gpm_Repeat

Declaration: `function Gpm_Repeat(millisecond:longint):longint;cdecl;external;`

Description: `Gpm_Repeat` returns 1 if no mouse event arrives in the next `millisecond` milliseconds, it returns 0 otherwise.

Errors: None.

See also: `Gpm_GetEvent` ([72](#))

Gpm_StrictDouble

Declaration: `function Gpm_StrictDouble(EventType : longint) : boolean;`

Description: `Gpm_StrictDouble` returns true if `EventType` contains only a doubleclick event, False otherwise.

Errors: None.

See also: `Gpm_StrictSingle` ([75](#)), `Gpm_AnyTriple` ([71](#)), `Gpm_AnyDouble` ([70](#)), `Gpm_StrictTriple` ([75](#)), `Gpm_AnySingle` ([71](#))

Gpm_StrictSingle

Declaration: `function Gpm_StrictSingle(EventType : longint) : boolean;`

Description: `Gpm_StrictSingle` returns True if `EventType` contains only a singleclick event, False otherwise.

Errors: None.

See also: `Gpm_AnyTriple` ([71](#)), `Gpm_StrictDouble` ([75](#)), `Gpm_AnyDouble` ([70](#)), `Gpm_StrictTriple` ([75](#)), `Gpm_AnySingle` ([71](#))

Gpm_StrictTriple

Declaration: `function Gpm_StrictTriple(EventType : longint) : boolean;`

Description: `Gpm_StrictTriple` returns true if `EventType` contains only a triple click event, False otherwise.

Errors: None.

See also: `Gpm_AnyTriple` ([71](#)), `Gpm_StrictDouble` ([75](#)), `Gpm_AnyDouble` ([70](#)), `Gpm_StrictSingle` ([75](#)), `Gpm_AnySingle` ([71](#))

Chapter 7

The GO32 unit

This chapter of the documentation describe the GO32 unit for the Free Pascal compiler under DOS. It was donated by Thomas Schatzl (tom_at_work@geocities.com), for which my thanks. This unit was first written for DOS by Florian Klaempfl. This chapter is divided in four sections. The first two sections are an introduction to the GO32 unit. The third section lists the pre-defined constants, types and variables. The last section describes the functions which appear in the interface part of the GO32 unit.

7.1 Introduction

These docs contain information about the GO32 unit. Only the GO32V2 DPMI mode is discussed by me here due to the fact that new applications shouldn't be created with the older GO32V1 model. The go32v2 version is much more advanced and better. Additionally a lot of functions only work in DPMI mode anyway. I hope the following explanations and introductions aren't too confusing at all. If you notice an error or bug send it to the FPC mailing list or directly to me. So let's get started and happy and error free coding I wish you....

Thomas Schatzl, 25. August 1998

7.2 Protected mode memory organization

What is DPMI

The DOS Protected Mode Interface helps you with various aspects of protected mode programming. These are roughly divided into descriptor handling, access to DOS memory, management of interrupts and exceptions, calls to real mode functions and other stuff. Additionally it automatically provides swapping to disk for memory intensive applications. A DPMI host (either a Windows DOS box or CWSDPMI.EXE) provides these functions for your programs.

Selectors and descriptors

Descriptors are a bit like real mode segments; they describe (as the name implies) a memory area in protected mode. A descriptor contains information about segment length, its base address and the attributes of it (i.e. type, access rights, ...). These descriptors are stored internally in a so-called descriptor table, which is basically an array of such descriptors. Selectors are roughly an index into this table. Because these 'segments' can be up to 4 GB in size, 32 bits aren't sufficient anymore to describe a single memory location like in real mode. 48 bits are now needed to do this, a 32 bit address and a 16 bit sized selector. The GO32 unit provides the tseginfo record to store such a

pointer. But due to the fact that most of the time data is stored and accessed in the %ds selector, FPC assumes that all pointers point to a memory location of this selector. So a single pointer is still only 32 bits in size. This value represents the offset from the data segment base address to this memory location.

FPC specialities

The %ds and %es selector **MUST** always contain the same value or some system routines may crash when called. The %fs selector is preloaded with the DOSMEMSELECTOR variable at startup, and it **MUST** be restored after use, because again FPC relies on this for some functions. Luckily we asm programmers can still use the %gs selector for our own purposes, but for how long ? See also: [get_cs \(92\)](#), [get_ds \(92\)](#), [gett_ss \(99\)](#), [allocate_ldt_descriptors \(85\)](#), [free_ldt_descriptor \(91\)](#), [segment_to_descriptor \(105\)](#), [get_next_selector_increment_value \(94\)](#), [get_segment_base_address \(98\)](#), [set_segment_base_address \(108\)](#), [set_segment_limit \(108\)](#), [create_code_segment_alias_descriptor \(88\)](#)

DOS memory access

DOS memory is accessed by the predefined `dosmemselector` selector; the GO32 unit additionally provides some functions to help you with standard tasks, like copying memory from heap to DOS memory and the likes. Because of this it is strongly recommended to use them, but you are still free to use the provided standard memory accessing functions which use 48 bit pointers. The third, but only thought for compatibility purposes, is using the `mem[]`-arrays. These arrays map the whole 1 Mb DOS space. They shouldn't be used within new programs. To convert a segment:offset real mode address to a protected mode linear address you have to multiply the segment by 16 and add its offset. This linear address can be used in combination with the DOSMEMSELECTOR variable. See also: [dosmemget \(89\)](#), [dosmemput \(90\)](#), [dosmemmove \(90\)](#), [dosmemfillchar \(88\)](#), [dosmemfillword \(89\)](#), `mem[]`-arrays, [seg_move \(106\)](#), [seg_fillchar \(104\)](#), [seg_fillword \(105\)](#).

I/O port access

The I/O port access is done via the various [inportb \(101\)](#), [outportb \(103\)](#) functions which are available. Additionally Free Pascal supports the Turbo Pascal `PORT[]`-arrays but it is by no means recommended to use them, because they're only for compatibility purposes. See also: [outportb \(103\)](#), [inportb \(101\)](#), `PORT[]`-arrays

Processor access

These are some functions to access various segment registers (%cs, %ds, %ss) which makes your work a bit easier. See also: [get_cs \(92\)](#), [get_ds \(92\)](#), [get_ss \(99\)](#)

Interrupt redirection

Interrupts are program interruption requests, which in one or another way get to the processor; there's a distinction between software and hardware interrupts. The former are explicitly called by an 'int' instruction and are a bit comparable to normal functions. Hardware interrupts come from external devices like the keyboard or mouse. Functions that handle hardware interrupts are called handlers.

Handling interrupts with DPMI

The interrupt functions are real-mode procedures; they normally can't be called in protected mode without the risk of an protection fault. So the DPMI host creates an interrupt descriptor table for the application. Initially all software interrupts (except for int 31h, 2Fh and 21h function 4Ch) or external hardware interrupts are simply directed to a handler that reflects the interrupt in real-mode, i.e. the DPMI host's default handlers switch the CPU to real-mode, issue the interrupt and switch back to protected mode. The contents of general registers and flags are passed to the real mode handler and the modified registers and flags are returned to the protected mode handler. Segment registers and stack pointer are not passed between modes.

Protected mode interrupts vs. Real mode interrupts

As mentioned before, there's a distinction between real mode interrupts and protected mode interrupts; the latter are protected mode programs, while the former must be real mode programs. To call a protected mode interrupt handler, an assembly 'int' call must be issued, while the other is called via the realintr() or intr() function. Consequently, a real mode interrupt then must either reside in DOS memory (<1MB) or the application must allocate a real mode callback address via the get_rm_callback() function.

Creating own interrupt handlers

Interrupt redirection with FPC pascal is done via the set_pm_interrupt() for protected mode interrupts or via the set_rm_interrupt() for real mode interrupts.

Disabling interrupts

The GO32 unit provides the two procedures disable() and enable() to disable and enable all interrupts.

Hardware interrupts

Hardware interrupts are generated by hardware devices when something unusual happens; this could be a keypress or a mouse move or any other action. This is done to minimize CPU time, else the CPU would have to check all installed hardware for data in a big loop (this method is called 'polling') and this would take much time. A standard IBM-PC has two interrupt controllers, that are responsible for these hardware interrupts: both allow up to 8 different interrupt sources (IRQs, interrupt requests). The second controller is connected to the first through IRQ 2 for compatibility reasons, e.g. if controller 1 gets an IRQ 2, he hands the IRQ over to controller 2. Because of this up to 15 different hardware interrupt sources can be handled. IRQ 0 through IRQ 7 are mapped to interrupts 8h to Fh and the second controller (IRQ 8 to 15) is mapped to interrupt 70h to 77h. All of the code and data touched by these handlers MUST be locked (via the various locking functions) to avoid page faults at interrupt time. Because hardware interrupts are called (as in real mode) with interrupts disabled, the handler has to enable them before it returns to normal program execution. Additionally a hardware interrupt must send an EOI (end of interrupt) command to the responsible controller; this is accomplished by sending the value 20h to port 20h (for the first controller) or A0h (for the second controller). The following example shows how to redirect the keyboard interrupt.

```
{ $ASMMODE ATT }  
{ $MODE FPC }
```

uses

```
  crt ,  
  go32 ;
```

```
const
    kbdint = $9;

var
    oldint9_handler : tseginfo;
    newint9_handler : tseginfo;

    clickproc : pointer;
    backupDS : Word; external name '___v2prt0_ds_alias';

procedure int9_handler; assembler;
asm
    cli
    pushl %ds
    pushl %es
    pushl %fs
    pushl %gs
    pushal
    movw %cs:backupDS, %ax
    movw %ax, %ds
    movw %ax, %es
    movw dosmemselector, %ax
    movw %ax, %fs
    call *clickproc
    popal
    popl %gs
    popl %fs
    popl %es
    popl %ds
    ljmp %cs:oldint9_handler
end;
procedure int9_dummy; begin end;

procedure clicker;
begin
    sound(500); delay(10); nosound;
end;
procedure clicker_dummy; begin end;

procedure install_click;
begin
    clickproc := @clicker;
    lock_data(clickproc, sizeof(clickproc));
    lock_data(dosmemselector, sizeof(dosmemselector));

    lock_code(@clicker,
        longint(@clicker_dummy) - longint(@clicker));
    lock_code(@int9_handler,
        longint(@int9_dummy) - longint(@int9_handler));
    newint9_handler.offset := @int9_handler;
    newint9_handler.segment := get_cs;
    get_pm_interrupt(kbdint, oldint9_handler);
    set_pm_interrupt(kbdint, newint9_handler);
end;

procedure remove_click;
begin
```



```
set_pm_interrupt(kbdint, oldint9_handler);
unlock_data(dosmemselector, sizeof(dosmemselector));
unlock_data(clickproc, sizeof(clickproc));

unlock_code(@clicker,
            longint(@clicker_dummy)-longint(@clicker));
unlock_code(@int9_handler,
            longint(@int9_dummy)-longint(@int9_handler));
end;

var
    ch : char;

begin
    install_click;
    WriteLn('Enter any message. Press return when finished');
    while (ch <> #13) do begin
        ch := readkey; write(ch);
    end;
    remove_click;
end.
```

Software interrupts

Ordinarily, a handler installed with `set_pm_interrupt` (106) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by `set_rm_interrupt` (107). See also `set_rm_interrupt` (107), `get_rm_interrupt` (97), `set_pm_interrupt` (106), `get_pm_interrupt` (94), `lock_data` (102), `lock_code` (102), `enable` (90), `disable` (88), `outportb` (103) Executing software interrupts Simply execute a `realintr()` call with the desired interrupt number and the supplied register data structure. But some of these interrupts require you to supply them a pointer to a buffer where they can store data to or obtain data from in memory. These interrupts are real mode functions and so they only can access the first Mb of linear address space, not FPC's data segment. For this reason FPC supplies a pre-initialized DOS memory location within the GO32 unit. This buffer is internally used for DOS functions too and so it's contents may change when calling other procedures. It's size can be obtained with `tb_size` (108) and it's linear address via `transfer_buffer` (109). Another way is to allocate a completely new DOS memory area via the `global_dos_alloc` (99) function for your use and supply its real mode address. See also: `tb_size` (108), `transfer_buffer` (109), `global_dos_alloc` (99), `global_dos_free` (101), `realintr` (104) The following examples illustrate the use of software interrupts.

```
uses
    go32;

var
    r : trealregs;

begin
    r.ah := $30;
    r.al := $01;
    realintr($21, r);
    WriteLn('DOS v', r.al, '.', r.ah, ' detected');
end.
```

```
uses
    crt,
    go32;
```

```
var
    r : trealregs;
    axreg : Word;

    oldint21h : tseginfo;
    newint21h : tseginfo;
procedure int21h_handler; assembler;
asm
    cmpw $0x3001, %ax
    jne .LCallOld
    movw $0x3112, %ax
    iret

.LCallOld:
    ljmp %cs:oldint21h
end;

procedure resume;
begin
    Writeln;
    Write('— press any key to resume —'); readkey;
    gotoxy(1, wherey); clreol;
end;

begin
    clrscr;
    Writeln('Executing real mode interrupt');
    resume;
    r.ah := $30; r.al := $01; realintr($21, r);
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
    resume;
    Writeln('Executing protected mode interrupt without our own',
        ' handler');
    Writeln;
    asm
        movb $0x30, %ah
        movb $0x01, %al
        int $0x21
        movw %ax, axreg
    end;
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
    resume;
    Writeln('As you can see the DPMI hosts default protected mode',
        ' handler');
    Writeln('simply redirects it to the real mode handler');
    resume;
    Writeln('Now exchanging the protected mode interrupt with our ',
        ' own handler');
    resume;

    newint21h.offset := @int21h_handler;
    newint21h.segment := get_cs;
    get_pm_interrupt($21, oldint21h);
    set_pm_interrupt($21, newint21h);

    Writeln('Executing real mode interrupt again');
    resume;
```

```
    r.ah := $30; r.al := $01; realintr($21, r);
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
    Writeln;
    Writeln('See, it didn''t change in any way. ');
    resume;
    Writeln('Now calling protected mode interrupt');
    resume;
    asm
        movb $0x30, %ah
        movb $0x01, %al
        int $0x21
        movw %ax, axreg
    end;
    Writeln('DOS v', Io(axreg), '.', hi(axreg), ' detected');
    Writeln;
    Writeln('Now you can see that there''s a distinction between ',
        'the two ways of calling interrupts... ');
    set_pm_interrupt($21, oldint21h);
end.
```

Real mode callbacks

The callback mechanism can be thought of as the converse of calling a real mode procedure (i.e. interrupt), which allows your program to pass information to a real mode program, or obtain services from it in a manner that's transparent to the real mode program. In order to make a real mode callback available, you must first get the real mode callback address of your procedure and the selector and offset of a register data structure. This real mode callback address (this is a segment:offset address) can be passed to a real mode program via a software interrupt, a DOS memory block or any other convenient mechanism. When the real mode program calls the callback (via a far call), the DPMI host saves the registers contents in the supplied register data structure, switches into protected mode, and enters the callback routine with the following settings:

- interrupts disabled
- %CS:%EIP = 48 bit pointer specified in the original call to `get_rm_callback` (95)
- %DS:%ESI = 48 bit pointer to real mode SS:SP
- %ES:%EDI = 48 bit pointer of real mode register data structure.
- %SS:%ESP = locked protected mode stack
- All other registers undefined

The callback procedure can then extract its parameters from the real mode register data structure and/or copy parameters from the real mode stack to the protected mode stack. Recall that the segment register fields of the real mode register data structure contain segment or paragraph addresses that are not valid in protected mode. Far pointers passed in the real mode register data structure must be translated to virtual addresses before they can be used with a protected mode program. The callback procedure exits by executing an IRET with the address of the real mode register data structure in %ES:%EDI, passing information back to the real mode caller by modifying the contents of the real mode register data structure and/or manipulating the contents of the real mode stack. The callback procedure is responsible for setting the proper address for resumption of real mode execution into the real mode register data structure; typically, this is accomplished by extracting the return address from the real mode stack and placing it into the %CS:%EIP fields of the real mode register data structure. After the IRET, the DPMI host switches the CPU back into real mode, loads ALL registers with the

contents of the real mode register data structure, and finally returns control to the real mode program. All variables and code touched by the callback procedure **MUST** be locked to prevent page faults. See also: [get_rm_callback \(95\)](#), [free_rm_callback \(91\)](#), [lock_code \(102\)](#), [lock_data \(102\)](#)

7.3 Types, Variables and Constants

Constants

Constants returned by `get_run_mode`

Tells you under what memory environment (e.g. memory manager) the program currently runs.

```
rm_unknown = 0; { unknown }
rm_raw     = 1; { raw (without HIMEM) }
rm_xms     = 2; { XMS (for example with HIMEM, without EMM386) }
rm_vcpi    = 3; { VCPI (for example HIMEM and EMM386) }
rm_dpml    = 4; { DPMI (for example \dos box or 386Max) }
```

Note: GO32V2 *always* creates DPMI programs, so you need a suitable DPMI host like CWS-DPMI.EXE or a Windows DOS box. So you don't need to check it, these constants are only useful in GO32V1 mode.

Processor flags constants

They are provided for a simple check with the flags identifier in the `trearegs` type. To check a single flag, simply do an AND operation with the flag you want to check. It's set if the result is the same as the flag value.

```
const carryflag = $001;
parityflag      = $004;
auxcarryflag    = $010;
zeroflag        = $040;
signflag        = $080;
trapflag        = $100;
interruptflag    = $200;
directionflag    = $400;
overflowflag     = $800;
```

Predefined types

```
type tmeminfo = record
    available_memory : Longint;
    available_pages  : Longint;
    available_lockable_pages : Longint;
    linear_space     : Longint;
    unlocked_pages   : Longint;
    available_physical_pages : Longint;
    total_physical_pages : Longint;
    free_linear_space : Longint;
    max_pages_in_paging_file : Longint;
    reserved : array[0..2] of Longint;
end;
```

Holds information about the memory allocation, etc. NOTE: The value of a field is -1 (0ffffffh) if

Table 7.1: Record description

Record entry	Description
available_memory	Largest available free block in bytes.
available_pages	Maximum unlocked page allocation in pages
available_lockable_pages	Maximum locked page allocation in pages.
linear_space	Linear address space size in pages.
unlocked_pages	Total number of unlocked pages.
available_physical_pages	Total number of free pages.
total_physical_pages	Total number of physical pages.
free_linear_space	Free linear address space in pages.
max_pages_in_paging_file	Size of paging file/partition in pages.

the value is unknown, it's only guaranteed, that available_memory contains a valid value. The size of the pages can be determined by the get_page_size() function.

```
type
trealregs = record
  case Integer of
    1: { 32-bit }
      (EDI, ESI, EBP, Res, EBX, EDX, ECX, EAX: Longint;
       Flags, ES, DS, FS, GS, IP, CS, SP, SS: Word);
    2: { 16-bit }
      (DI, DI2, SI, SI2, BP, BP2, R1, R2: Word;
       BX, BX2, DX, DX2, CX, CX2, AX, AX2: Word);
    3: { 8-bit }
      (stuff: array[1..4] of Longint;
       BL, BH, BL2, BH2, DL, DH, DL2, DH2, CL,
       CH, CL2, CH2, AL, AH, AL2, AH2: Byte);
    4: { Compat }
      (RealeDI, RealeSI, RealeBP, RealRES, RealeBX,
       RealeDX, RealeCX, RealeAX: Longint;
       RealFlags, RealeS, RealDS, RealFS, RealGS,
       RealIP, RealCS, RealSP, RealSS: Word);
  end;
registers = trealregs;
```

These two types contain the data structure to pass register values to a interrupt handler or real mode callback.

```
type tseginfo = record
  offset : Pointer; segment : Word; end;
```

This record is used to store a full 48-bit pointer. This may be either a protected mode selector:offset address or in real mode a segment:offset address, depending on application. See also: Selectors and descriptors, DOS memory access, Interrupt redirection

Variables.

```
var dosmemselector : Word;
```

Selector to the DOS memory. The whole DOS memory is automatically mapped to this single descriptor at startup. This selector is the recommended way to access DOS memory.

```
var int31error : Word;
```

This variable holds the result of a DPMI interrupt call. Any nonzero value must be treated as a critical failure.

7.4 Functions and Procedures

allocate_ldt_descriptors

Declaration: Function `allocate_ldt_descriptors` (`count` : `Word`) : `Word`;

Description: Allocates a number of new descriptors. Parameters:

count: specifies the number of requested unique descriptors.

Return value: The base selector. Notes: The descriptors allocated must be initialized by the application with other function calls. This function returns descriptors with a limit and size value set to zero. If more than one descriptor was requested, the function returns a base selector referencing the first of a contiguous array of descriptors. The selector values for subsequent descriptors in the array can be calculated by adding the value returned by the `get_next_selector_increment_value` (94) function.

Errors: Check the `int31error` variable.

See also: `free_ldt_descriptor` (91), `get_next_selector_increment_value` (94), `segment_to_descriptor` (105), `create_code_segment_alias_descriptor` (88), `set_segment_limit` (108), `set_segment_base_address` (108)

```
{ $mode delphi }
uses
    crt,
    go32;

const
    maxx = 80;
    maxy = 25;
    bytespercell = 2;
    screensize = maxx * maxy * bytespercell;

    linB8000 = $B800 * 16;

type
    string80 = string[80];

var
    text_save : array[0..screensize-1] of byte;
    text_oldx, text_oldy : Word;

    text_sel : Word;

procedure status(s : string80);
begin
    gotoxy(1, 1); clrscr; write(s); readkey;
end;
```

```
procedure selinfo(sel : Word);
begin
  gotoxy(1, 24);
  clreol; writeln('Descriptor base address : $',
    hexstr(get_segment_base_address(sel), 8));
  clreol; write('Descriptor limit : ', get_segment_limit(sel));
end;

function makechar(ch : char; color : byte) : Word;
begin
  result := byte(ch) or (color shl 8);
end;

begin
  seg_move(dosmemselector, linB8000, get_ds, longint(@text_save),
    screensize);
  text_oldx := wherex; text_oldy := wherey;
  seg_fillword(dosmemselector, linB8000, screensize div 2,
    makechar(' ', Black or (Black shl 4)));
  status('Creating selector ''text_sel'' to a part of ' +
    'text screen memory');
  text_sel := allocate_ldt_descriptors(1);
  set_segment_base_address(text_sel,
    linB8000 + bytespercell * maxx * 1);
  set_segment_limit(text_sel, screensize - 1 - bytespercell *
    maxx * 3);
  selinfo(text_sel);

  status('and clearing entire memory selected by ''text_sel'' +
    ' descriptor');
  seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
    makechar(' ', LightBlue shl 4));

  status('Notice that only the memory described by the ' +
    ' descriptor changed, nothing else');

  status('Now reducing it''s limit and base and setting it''s ' +
    'described memory');
  set_segment_base_address(text_sel,
    get_segment_base_address(text_sel) + bytespercell * maxx);
  set_segment_limit(text_sel,
    get_segment_limit(text_sel) - bytespercell * maxx * 2);
  selinfo(text_sel);
  status('Notice that the base addr increased by one line but ' +
    'the limit decreased by 2 lines');
  status('This should give you the hint that the limit is ' +
    'relative to the base');
  seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
    makechar(#176, LightMagenta or Brown shl 4));

  status('Now let''s get crazy and copy 10 lines of data from ' +
    'the previously saved screen');
  seg_move(get_ds, longint(@text_save), text_sel,
    maxx * bytespercell * 2, maxx * bytespercell * 10);

  status('At last freeing the descriptor and restoring the old ' +
    'screen contents..');
  status('I hope this little program may give you some hints on ' +
```

```
    'working with descriptors');  
    free_ldt_descriptor(text_sel);  
    seg_move(get_ds, longint(@text_save), dosmemselector,  
            linB8000, screensize);  
    gotoxy(text_oldx, text_oldy);  
end.
```

allocate_memory_block

Declaration: `Function allocate_memory_block (size:Longint) : Longint;`

Description: Allocates a block of linear memory. Parameters:

size: Size of requested linear memory block in bytes.

Returned values: blockhandle - the memory handle to this memory block. Linear address of the requested memory. Notes: **WARNING:** According to my DPMI docs this function is not implemented correctly. Normally you should also get a blockhandle to this block after successful operation. This handle can then be used to free the memory block afterwards or use this handle for other purposes. Since the function isn't implemented correctly, and doesn't return a blockhandle, the block can't be deallocated and is hence unusable ! This function doesn't allocate any descriptors for this block, it's the applications responsibility to allocate and initialize for accessing this memory.

Errors: Check the `int31error` variable.

See also: `free_memory_block` ([91](#))

copyfromdos

Declaration: `Procedure copyfromdos (var addr; len : Longint);`

Description: Copies data from the pre-allocated DOS memory transfer buffer to the heap. Parameters:

addr: data to copy to.

len: number of bytes to copy to heap.

Notes: Can only be used in conjunction with the DOS memory transfer buffer.

Errors: Check the `int31error` variable.

See also: `tb_size` ([108](#)), `transfer_buffer` ([109](#)), `copytodos` ([87](#))

copytodos

Declaration: `Procedure copytodos (var addr; len : Longint);`

Description: Copies data from heap to the pre-allocated DOS memory buffer. Parameters:

addr: data to copy from.

len: number of bytes to copy to DOS memory buffer.

Notes: This function fails if you try to copy more bytes than the transfer buffer is in size. It can only be used in conjunction with the transfer buffer.

Errors: Check the `int31error` variable.

See also: `tb_size` ([108](#)), `transfer_buffer` ([109](#)), `copyfromdos` ([87](#))

create_code_segment_alias_descriptor

Declaration: `Function create_code_segment_alias_descriptor (seg : Word) : Word;`

Description: Creates a new descriptor that has the same base and limit as the specified descriptor. Parameters:

seg: Descriptor.

Return values: The data selector (alias). Notes: In effect, the function returns a copy of the descriptor. The descriptor alias returned by this function will not track changes to the original descriptor. In other words, if an alias is created with this function, and the base or limit of the original segment is then changed, the two descriptors will no longer map the same memory.

Errors: Check the `int31error` variable.

See also: `allocate_ldt_descriptors` (85), `set_segment_limit` (108), `set_segment_base_address` (108)

disable

Declaration: `Procedure disable ;`

Description: Disables all hardware interrupts by execution a CLI instruction. Parameters: None.

Errors: None.

See also: `enable` (90)

dosmemfillchar

Declaration: `Procedure dosmemfillchar (seg, ofs : Word; count : Longint; c : char);`

Description: Sets a region of DOS memory to a specific byte value. Parameters:

seg: real mode segment.

ofs: real mode offset.

count: number of bytes to set.

c: value to set memory to.

Notes: No range check is performed.

Errors: None.

See also: `dosmemput` (90), `dosmemget` (89), `dosmemmove` (90), `dosmemmove`, `dosmemfillword` (89), `seg_move` (106), `seg_fillchar` (104), `seg_fillword` (105)

uses

`crt ,
go32 ;`

const

`columns = 80;
rows = 25;
screensize = rows*columns*2;`

`text = '! Hello world !';`

var

`textofs : Longint;`

```
    save_screen : array[0..screensize-1] of byte;  
    curx, cury : Integer;  
  
begin  
    randomize;  
    dosmemget($B800, 0, save_screen, screensize);  
    curx := wherex; cury := wherey;  
    gotoxy(1, 1); Write(text);  
    textofs := screensize + length(text)*2;  
    dosmemmove($B800, 0, $B800, textofs, length(text)*2);  
    dosmemfillchar($B800, 0, screensize, #0);  
    while (not keypressed) do begin  
        dosmemfillchar($B800, textofs + random(length(text))*2 + 1,  
            1, char(random(255)));  
        dosmemmove($B800, textofs, $B800,  
            random(columns)*2+random(rows)*columns*2,  
            length(text)*2);  
        delay(1);  
    end;  
    readkey;  
    readkey;  
    dosmemput($B800, 0, save_screen, screensize);  
    gotoxy(curx, cury);  
end.
```

dosmemfillword

Declaration: Procedure dosmemfillword (seg, ofs : Word; count : Longint; w : Word);

Description: Sets a region of DOS memory to a specific word value. Parameters:

seg: real mode segment.

ofs: real mode offset.

count: number of words to set.

w: value to set memory to.

Notes: No range check is performed.

Errors: None.

See also: dosmemput (90), dosmemget (89), dosmemmove (90), dosmemfillchar (88), seg_move (106),
seg_fillchar (104), seg_fillword (105)

dosmemget

Declaration: Procedure dosmemget (seg : Word; ofs : Word; var data; count : Longint);

Description: Copies data from the DOS memory onto the heap. Parameters:

seg: source real mode segment.

ofs: source real mode offset.

data: destination.

count: number of bytes to copy.

Notes: No range checking is performed.

Errors: None.

See also: [dosmempmut \(90\)](#), [dosmemmove \(90\)](#), [dosmemfillchar \(88\)](#), [dosmemfillword \(89\)](#), [seg_move \(106\)](#), [seg_fillchar \(104\)](#), [seg_fillword \(105\)](#)

For an example, see [global_dos_alloc \(99\)](#).

dosmemmove

Declaration: `Procedure dosmemmove (sseg, sofs, dseg, dofs : Word; count : Longint);`

Description: Copies count bytes of data between two DOS real mode memory locations. Parameters:

sseg: source real mode segment.

sofs: source real mode offset.

dseg: destination real mode segment.

dofs: destination real mode offset.

count: number of bytes to copy.

Notes: No range check is performed in any way.

Errors: None.

See also: [dosmempmut \(90\)](#), [dosmemget \(89\)](#), [dosmemfillchar \(88\)](#), [dosmemfillword \(89\)](#), [seg_move \(106\)](#), [seg_fillchar \(104\)](#), [seg_fillword \(105\)](#)

For an example, see [seg_fillchar \(104\)](#).

dosmempmut

Declaration: `Procedure dosmempmut (seg : Word; ofs : Word; var data; count : Longint);`

Description: Copies heap data to DOS real mode memory. Parameters:

seg: destination real mode segment.

ofs: destination real mode offset.

data: source.

count: number of bytes to copy.

Notes: No range checking is performed.

Errors: None.

See also: [dosmemget \(89\)](#), [dosmemmove \(90\)](#), [dosmemfillchar \(88\)](#), [dosmemfillword \(89\)](#), [seg_move \(106\)](#), [seg_fillchar \(104\)](#), [seg_fillword \(105\)](#)

For an example, see [global_dos_alloc \(99\)](#).

enable

Declaration: `Procedure enable ;`

Description: Enables all hardware interrupts by executing a STI instruction. Parameters: None.

Errors: None.

See also: [disable \(88\)](#)

free_ldt_descriptor

Declaration: `Function free_ldt_descriptor (des : Word) : boolean;`

Description: Frees a previously allocated descriptor. Parameters:

des: The descriptor to be freed.

Return value: `True` if successful, `False` otherwise. Notes: After this call this selector is invalid and must not be used for any memory operations anymore. Each descriptor allocated with `allocate_ldt_descriptors` (85) must be freed individually with this function, even if it was previously allocated as a part of a contiguous array of descriptors.

Errors: Check the `int31error` variable.

See also: `allocate_ldt_descriptors` (85), `get_next_selector_increment_value` (94)

For an example, see `allocate_ldt_descriptors` (85).

free_memory_block

Declaration: `Function free_memory_block (blockhandle : Longint) : boolean;`

Description: Frees a previously allocated memory block. Parameters:

blockhandle: the handle to the memory area to free.

Return value: `True` if successful, `false` otherwise. Notes: Frees memory that was previously allocated with `allocate_memory_block` (87). This function doesn't free any descriptors mapped to this block, it's the application's responsibility.

Errors: Check `int31error` variable.

See also: `allocate_memory_block` (87)

free_rm_callback

Declaration: `Function free_rm_callback (var intaddr : tseginfo) : boolean;`

Description: Releases a real mode callback address that was previously allocated with the `get_rm_callback` (95) function. Parameters:

intaddr: real mode address buffer returned by `get_rm_callback` (95).

Return values: `True` if successful, `False` if not

Errors: Check the `int31error` variable.

See also: `set_rm_interrupt` (107), `get_rm_callback` (95)

For an example, see `get_rm_callback` (95).

get_cs

Declaration: `Function get_cs : Word;`

Description: Returns the cs selector. Parameters: None. Return values: The content of the cs segment register.

Errors: None.

See also: [get_ds \(92\)](#), [get_ss \(99\)](#)

For an example, see [set_pm_interrupt \(106\)](#).

get_descriptor_access_rights

Declaration: `Function get_descriptor_access_rights (d : Word) : Longint;`

Description: Gets the access rights of a descriptor. Parameters:

`d` selector to descriptor.

Return value: Access rights bit field.

Errors: Check the `int31error` variable.

See also: [set_descriptor_access_rights \(106\)](#)

get_ds

Declaration: `Function get_ds : Word;`

Description: Returns the ds selector. Parameters: None. Return values: The content of the ds segment register.

Errors: None.

See also: [get_cs \(92\)](#), [get_ss \(99\)](#)

get_linear_addr

Declaration: `Function get_linear_addr (phys_addr : Longint; size : Longint) : Longint;`

Description: Converts a physical address into a linear address. Parameters:

phys_addr: physical address of device.

size: Size of region to map in bytes.

Return value: Linear address that can be used to access the physical memory. Notes: It's the applications responsibility to allocate and set up a descriptor for access to the memory. This function shouldn't be used to map real mode addresses.

Errors: Check the `int31error` variable.

See also: [allocate_ldt_descriptors \(85\)](#), [set_segment_limit \(108\)](#), [set_segment_base_address \(108\)](#)

get_meminfo

Declaration: Function `get_meminfo (var meminfo : tmeminfo) : boolean;`

Description: Returns information about the amount of available physical memory, linear address space, and disk space for page swapping. Parameters:

meminfo: buffer to fill memory information into.

Return values: Due to an implementation bug this function always returns `False`, but it always succeeds. **Notes:** Only the first field of the returned structure is guaranteed to contain a valid value. Any fields that are not supported by the DPML host will be set by the host to `-1` (`0FFFFFFFFH`) to indicate that the information is not available. The size of the pages used by the DPML host can be obtained with the `get_page_size` (94) function.

Errors: Check the `int31error` variable.

See also: `get_page_size` (94)

```

uses
    go32;

var
    meminfo : tmeminfo;

begin
    get_meminfo(meminfo);
    if (int31error <> 0) then begin
        Writeln('Error getting DPML memory information... Halting');
        Writeln('DPML error number : ', int31error);
    end else begin
        with meminfo do begin
            Writeln('Largest available free block : ',
                available_memory div 1024, ' kbytes');
            if (available_pages <> -1) then
                Writeln('Maximum available unlocked pages : ',
                    available_pages);
            if (available_lockable_pages <> -1) then
                Writeln('Maximum lockable available pages : ',
                    available_lockable_pages);
            if (linear_space <> -1) then
                Writeln('Linear address space size : ',
                    linear_space*get_page_size div 1024, ' kbytes');
            if (unlocked_pages <> -1) then
                Writeln('Total number of unlocked pages : ',
                    unlocked_pages);
            if (available_physical_pages <> -1) then
                Writeln('Total number of free pages : ',
                    available_physical_pages);
            if (total_physical_pages <> -1) then
                Writeln('Total number of physical pages : ',
                    total_physical_pages);
            if (free_linear_space <> -1) then
                Writeln('Free linear address space : ',
                    free_linear_space*get_page_size div 1024,
                    ' kbytes');
            if (max_pages_in_paging_file <> -1) then
                Writeln('Maximum size of paging file : ',
                    max_pages_in_paging_file*get_page_size div 1024,

```

```
                                ' kbytes' );  
                                end;  
                                end;  
end.
```

get_next_selector_increment_value

Declaration: Function `get_next_selector_increment_value` : `Word`;

Description: Returns the selector increment value when allocating multiple subsequent descriptors via `allocate_ldt_descriptors` (85). Parameters: None. Return value: Selector increment value. Notes: Because `allocate_ldt_descriptors` (85) only returns the selector for the first descriptor and so the value returned by this function can be used to calculate the selectors for subsequent descriptors in the array.

Errors: Check the `int31error` variable.

See also: `allocate_ldt_descriptors` (85), `free_ldt_descriptor` (91)

get_page_size

Declaration: Function `get_page_size` : `Longint`;

Description: Returns the size of a single memory page. Return value: Size of a single page in bytes. Notes: The returned size is typically 4096 bytes.

Errors: Check the `int31error` variable.

See also: `get_meminfo` (93)

For an example, see `get_meminfo` (93).

get_pm_interrupt

Declaration: Function `get_pm_interrupt` (`vector` : `byte`; `var intaddr` : `tseginfo`) : `boolean`;

Description: Returns the address of a current protected mode interrupt handler. Parameters:

vector: interrupt handler number you want the address to.

intaddr: buffer to store address.

Return values: `True` if successful, `False` if not. Notes: The returned address is a protected mode selector:offset address.

Errors: Check the `int31error` variable.

See also: `set_pm_interrupt` (106), `set_rm_interrupt` (107), `get_rm_interrupt` (97)

For an example, see `set_pm_interrupt` (106).

get_rm_callback

Declaration: Function `get_rm_callback` (`pm_func` : pointer; `const reg` : `trealregs`; `var rmcb`: `tseginfo`) : `boolean`;

Description: Returns a unique real mode `segment:offset` address, known as a "real mode callback," that will transfer control from real mode to a protected mode procedure. Parameters:

pm_func: pointer to the protected mode callback function.

reg: supplied registers structure.

rmcb: buffer to real mode address of callback function.

Return values: `True` if successful, otherwise `False`. **Notes:** Callback addresses obtained with this function can be passed by a protected mode program for example to an interrupt handler, device driver, or TSR, so that the real mode program can call procedures within the protected mode program or notify the protected mode program of an event. The contents of the supplied `regs` structure is not valid after function call, but only at the time of the actual callback.

Errors: Check the `int31error` variable.

See also: `free_rm_callback` ([91](#))

```
{ $ASMMODE ATT }
{ $MODE FPC }

uses
    crt,
    go32;

const
    mouseint = $33;

var
    mouse_regs      : trealregs; external name '___v2prt0_rmcb_regs';
    mouse_seginfo   : tseginfo;

var
    mouse_numbuttons : longint;

    mouse_action    : word;
    mouse_x, mouse_y : Word;
    mouse_b         : Word;

    userproc_installed : Longbool;
    userproc_length   : Longint;
    userproc_proc     : pointer;

procedure callback_handler; assembler;
asm
    pushw %ds
    pushl %eax
    movw %es, %ax
    movw %ax, %ds

    cmpl $1, USERPROC_INSTALLED
    jne .LNoCallback
    pushal
    movw DOSmemSELECTOR, %ax
    movw %ax, %fs
```



```
    call *USERPROC_PROC
    popal
.LNoCallback:

    popl %eax
    popw %ds

    pushl %eax
    movl (%esi), %eax
    movl %eax, %es:42(%edi)
    addw $4, %es:46(%edi)
    popl %eax
    iret
end;
procedure mouse_dummy; begin end;

procedure textuserproc;
begin
    mouse_b := mouse_regs.bx;
    mouse_x := (mouse_regs.cx shr 3) + 1;
    mouse_y := (mouse_regs.dx shr 3) + 1;
end;

procedure install_mouse(userproc : pointer; userproclen : longint);
var r : trealregs;
begin
    r.eax := $0; realintr(mouseint, r);
    if (r.eax <> $FFFF) then begin
        Writeln('No Microsoft compatible mouse found');
        Writeln('A Microsoft compatible mouse driver is necessary ',
            'to run this example');
        halt;
    end;
    if (r.bx = $ffff) then mouse_numbuttons := 2
    else mouse_numbuttons := r.bx;
    Writeln(mouse_numbuttons, ' button Microsoft compatible mouse ',
        ' found. ');
    if (userproc <> nil) then begin
        userproc_proc := userproc;
        userproc_installed := true;
        userproc_length := userproclen;
        lock_code(userproc_proc, userproc_length);
    end else begin
        userproc_proc := nil;
        userproc_length := 0;
        userproc_installed := false;
    end;
    lock_data(mouse_x, sizeof(mouse_x));
    lock_data(mouse_y, sizeof(mouse_y));
    lock_data(mouse_b, sizeof(mouse_b));
    lock_data(mouse_action, sizeof(mouse_action));

    lock_data(userproc_installed, sizeof(userproc_installed));
    lock_data(userproc_proc, sizeof(userproc_proc));

    lock_data(mouse_regs, sizeof(mouse_regs));
    lock_data(mouse_seginf, sizeof(mouse_seginf));
    lock_code(@callback_handler,
```

```
        longint(@mouse_dummy)-longint(@callback_handler));
get_rm_callback(@callback_handler, mouse_regs, mouse_seginfo);
r.eax := $0c; r.ecx := $7f;
r.edx := longint(mouse_seginfo.offset);
r.es := mouse_seginfo.segment;
realintr(mouseint, r);
r.eax := $01;
realintr(mouseint, r);
end;

procedure remove_mouse;
var
    r : trealregs;
begin
    r.eax := $02; realintr(mouseint, r);
    r.eax := $0c; r.ecx := 0; r.edx := 0; r.es := 0;
    realintr(mouseint, r);
    free_rm_callback(mouse_seginfo);
    if (userproc_installed) then begin
        unlock_code(userproc_proc, userproc_length);
        userproc_proc := nil;
        userproc_length := 0;
        userproc_installed := false;
    end;
    unlock_data(mouse_x, sizeof(mouse_x));
    unlock_data(mouse_y, sizeof(mouse_y));
    unlock_data(mouse_b, sizeof(mouse_b));
    unlock_data(mouse_action, sizeof(mouse_action));

    unlock_data(userproc_proc, sizeof(userproc_proc));
    unlock_data(userproc_installed, sizeof(userproc_installed));

    unlock_data(mouse_regs, sizeof(mouse_regs));
    unlock_data(mouse_seginfo, sizeof(mouse_seginfo));
    unlock_code(@callback_handler,
        longint(@mouse_dummy)-longint(@callback_handler));
    fillchar(mouse_seginfo, sizeof(mouse_seginfo), 0);
end;

begin
    install_mouse(@textuserproc, 400);
    Writeln('Press any key to exit...');
    while (not keypressed) do begin
        gotoxy(1, wherey);
        write('MouseX : ', mouse_x:2, ' MouseY : ', mouse_y:2,
            ' Buttons : ', mouse_b:2);
    end;
    remove_mouse;
end.
```

get_rm_interrupt

Declaration: Function get_rm_interrupt (vector : byte; var intaddr : tseginfo)
: boolean;

Description: Returns the contents of the current machine's real mode interrupt vector for the specified interrupt.

Parameters:

vector: interrupt vector number.

intaddr: buffer to store real mode segment:offset address.

Return values: True if successful, False otherwise. Notes: The returned address is a real mode segment address, which isn't valid in protected mode.

Errors: Check the `int31error` variable.

See also: `set_rm_interrupt` (107), `set_pm_interrupt` (106), `get_pm_interrupt` (94)

get_run_mode

Declaration: `Function get_run_mode : Word;`

Description: Returns the current mode your application runs with. Return values: One of the constants used by this function.

Errors: None.

See also: constants returned by `get_run_mode` (98)

```
uses
    go32;

begin
    case (get_run_mode) of
        rm_unknown :
            Writeln('Unknown environment found');
        rm_raw :
            Writeln('You are currently running in raw mode ',
                '(without HIMEM)');
        rm_xms :
            Writeln('You are currently using HIMEM.SYS only');
        rm_vcpi :
            Writeln('VCPI server detected. You're using HIMEM and ',
                'EMM386');
        rm_dpml :
            Writeln('DPMI detected. You're using a DPMI host like ',
                'a windows DOS box or CWSDPMI');
    end;
end.
```

get_segment_base_address

Declaration: `Function get_segment_base_address (d : Word) : Longint;`

Description: Returns the 32-bit linear base address from the descriptor table for the specified segment. Parameters:

d: selector of the descriptor you want the base address of.

Return values: Linear base address of specified descriptor.

Errors: Check the `int31error` variable.

See also: [allocate_ldt_descriptors \(85\)](#), [set_segment_base_address \(108\)](#), [allocate_ldt_descriptors \(85\)](#), [set_segment_limit \(108\)](#), [get_segment_limit \(99\)](#)

For an example, see [allocate_ldt_descriptors \(85\)](#).

get_segment_limit

Declaration: `Function get_segment_limit (d : Word) : Longint;`

Description: Returns a descriptors segment limit. Parameters:

d: selector.

Return value: Limit of the descriptor in bytes.

Errors: Returns zero if descriptor is invalid.

See also: [allocate_ldt_descriptors \(85\)](#), [set_segment_limit \(108\)](#), [set_segment_base_address \(108\)](#), [get_segment_base_address \(98\)](#),

get_ss

Declaration: `Function get_ss : Word;`

Description: Returns the ss selector. Parameters: None. Return values: The content of the ss segment register.

Errors: None.

See also: [get_ds \(92\)](#), [get_cs \(92\)](#)

global_dos_alloc

Declaration: `Function global_dos_alloc (bytes : Longint) : Longint;`

Description: Allocates a block of DOS real mode memory. Parameters:

bytes: size of requested real mode memory.

Return values: The low word of the returned value contains the selector to the allocated DOS memory block, the high word the corresponding real mode segment value. The offset value is always zero. This function allocates memory from DOS memory pool, i.e. memory below the 1 MB boundary that is controlled by DOS. Such memory blocks are typically used to exchange data with real mode programs, TSRs, or device drivers. The function returns both the real mode segment base address of the block and one descriptor that can be used by protected mode applications to access the block. This function should only be used for temporary buffers to get real mode information (e.g. interrupts that need a data structure in ES:(E)DI), because every single block needs a unique selector. The returned selector should only be freed by a [global_dos_free \(101\)](#) call.

Errors: Check the `int31error` variable.

See also: [global_dos_free \(101\)](#)

uses

`go32;`

procedure `dosalloc(var selector : word;
var segment : word; size : longint);`

```
var
    res : longint;
begin
    res := global_dos_alloc(size);
    selector := word(res);
    segment := word(res shr 16);
end;

procedure dosfree(selector : word);
begin
    global_dos_free(selector);
end;

type
    VBEInfoBuf = packed record
        Signature : array[0..3] of char;
        Version : Word;
        reserved : array[0..505] of byte;
    end;

var
    selector,
    segment : Word;

    r : trealregs;
    infobuf : VBEInfoBuf;

begin
    fillchar(r, sizeof(r), 0);
    fillchar(infobuf, sizeof(VBEInfoBuf), 0);
    dosalloc(selector, segment, sizeof(VBEInfoBuf));
    if (int31error<>0) then begin
        Writeln('Error while allocating real mode memory, halting');
        halt;
    end;
    infobuf.Signature := 'VBE2';
    dosmempu(segment, 0, infobuf, sizeof(infobuf));
    r.ax := $4f00; r.es := segment;
    realintr($10, r);
    dosmemge(segment, 0, infobuf, sizeof(infobuf));
    dosfree(selector);
    if (r.ax <> $4f) then begin
        Writeln('VBE BIOS extension not available, function call ',
            'failed');
        halt;
    end;
    if (infobuf.signature[0] = 'V') and
        (infobuf.signature[1] = 'E') and
        (infobuf.signature[2] = 'S') and
        (infobuf.signature[3] = 'A') then begin
        Writeln('VBE version ', hi(infobuf.version), '.',
            lo(infobuf.version), ' detected');
    end;
end.
```

global_dos_free

Declaration: `Function global_dos_free (selector :Word) : boolean;`

Description: Frees a previously allocated DOS memory block. Parameters:

selector: selector to the DOS memory block.

Return value: `True` if successful, `False` otherwise. Notes: The descriptor allocated for the memory block is automatically freed and hence invalid for further use. This function should only be used for memory allocated by `global_dos_alloc` (99).

Errors: Check the `int31error` variable.

See also: `global_dos_alloc` (99)

For an example, see `global_dos_alloc` (99).

inportb

Declaration: `Function inportb (port : Word) : byte;`

Description: Reads 1 byte from the selected I/O port. Parameters:

port: the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: `outportb` (103), `inportw` (101), `inportl` (101)

inportl

Declaration: `Function inportl (port : Word) : Longint;`

Description: Reads 1 longint from the selected I/O port. Parameters:

port: the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: `outportb` (103), `inportb` (101), `inportw` (101)

inportw

Declaration: `Function inportw (port : Word) : Word;`

Description: Reads 1 word from the selected I/O port. Parameters:

port: the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: `outportw` (103), `inportb` (101), `inportl` (101)

lock_code

Declaration: `Function lock_code (functionaddr : pointer; size : Longint) : boolean;`

Description: Locks a memory range which is in the code segment selector. Parameters:

functionaddr: address of the function to be locked.

size: size in bytes to be locked.

Return values: True if successful, False otherwise.

Errors: Check the `int31error` variable.

See also: `lock_linear_region` (102), `lock_data` (102), `unlock_linear_region` (109), `unlock_data` (109), `unlock_code` (109)

For an example, see `get_rm_callback` (95).

lock_data

Declaration: `Function lock_data (var data; size : Longint) : boolean;`

Description: Locks a memory range which resides in the data segment selector. Parameters:

data: address of data to be locked.

size: length of data to be locked.

Return values: True if successful, False otherwise.

Errors: Check the `int31error` variable.

See also: `lock_linear_region` (102), `lock_code` (102), `unlock_linear_region` (109), `unlock_data` (109), `unlock_code` (109)

For an example, see `get_rm_callback` (95).

lock_linear_region

Declaration: `Function lock_linear_region (linearaddr, size : Longint) : boolean;`

Description: Locks a memory region to prevent swapping of it. Parameters:

linearaddr: the linear address of the memory are to be locked.

size: size in bytes to be locked.

Return value: True if successful, False otherwise.

Errors: Check the `int31error` variable.

See also: `lock_data` (102), `lock_code` (102), `unlock_linear_region` (109), `unlock_data` (109), `unlock_code` (109)

outportb

Declaration: `Procedure outportb (port : Word; data : byte);`

Description: Sends 1 byte of data to the specified I/O port. Parameters:

port: the I/O port number to send data to.

data: value sent to I/O port.

Return values: None.

Errors: None.

See also: [inportb \(101\)](#), [outportl \(103\)](#), [outportw \(103\)](#)

uses

`crt ,
go32 ;`

begin

`outportb ($61 , $ff);
delay (50);
outportb ($61 , $0);`

end .

outportl

Declaration: `Procedure outportl (port : Word; data : Longint);`

Description: Sends 1 longint of data to the specified I/O port. Parameters:

port: the I/O port number to send data to.

data: value sent to I/O port.

Return values: None.

Errors: None.

See also: [inportl \(101\)](#), [outportw \(103\)](#), [outportb \(103\)](#)

For an example, see [outportb \(103\)](#).

outportw

Declaration: `Procedure outportw (port : Word; data : Word);`

Description: Sends 1 word of data to the specified I/O port. Parameters:

port: the I/O port number to send data to.

data: value sent to I/O port.

Return values: None.

Errors: None.

See also: [inportw \(101\)](#), [outportl \(103\)](#), [outportb \(103\)](#)

For an example, see [outportb \(103\)](#).

realintr

Declaration: Function `realintr` (`intrnr`: Word; `var regs` : `trealregs`) : boolean;

Description: Simulates an interrupt in real mode. Parameters:

intrnr: interrupt number to issue in real mode.

regs: registers data structure.

Return values: The supplied registers data structure contains the values that were returned by the real mode interrupt. `True` if successful, `False` if not. Notes: The function transfers control to the address specified by the real mode interrupt vector of `intrnr`. The real mode handler must return by executing an `IRET`.

Errors: Check the `int31error` variable.

See also:

```
uses
    go32;

var
    r : trealregs;

begin
    r.ax := $5300;
    r.bx := 0;
    realintr($15, r);
    if ((r.flags and carryflag)=0) then begin
        Writeln('APM v', (r.ah and $f), '.',
                (r.al shr 4), (r.al and $f), ' detected');
    end else
        Writeln('APM not present');
end.
```

seg_fillchar

Declaration: Procedure `seg_fillchar` (`seg` : Word; `ofs` : Longint; `count` : Longint; `c` : char);

Description: Sets a memory area to a specific value. Parameters:

seg: selector to memory area.

ofs: offset to memory.

count: number of bytes to set.

c: byte data which is set.

Return values: None. Notes: No range check is done in any way.

Errors: None.

See also: `seg_move` ([106](#)), `seg_fillword` ([105](#)), `dosmemfillchar` ([88](#)), `dosmemfillword` ([89](#)), `dosmemget` ([89](#)), `dosmempu` ([90](#)), `dosmemmove` ([90](#))

```
uses
    go32;
```

```
var
    vgasel : Word;
    r : trealregs;

begin
    r.eax := $13; realintr($10, r);
    vgasel := segment_to_descriptor($A000);
    seg_fillchar(vgasel, 0, 64000, #15);
    readln;
    r.eax := $3; realintr($10, r);
end.
```

seg_fillword

Declaration: Procedure seg_fillword (seg : Word; ofs : Longint; count : Longint; w : Word);

Description: Sets a memory area to a specific value. Parameters:

seg: selector to memory area.

ofs: offset to memory.

count: number of words to set.

w: word data which is set.

Return values: None. Notes: No range check is done in any way.

Errors: None.

See also: seg_move ([106](#)), seg_fillchar ([104](#)), dosmemfillchar ([88](#)), dosmemfillword ([89](#)), dosmemget ([89](#)), dosmemput ([90](#)), dosmemmove ([90](#))

For an example, see allocate_ldt_descriptors ([85](#)).

segment_to_descriptor

Declaration: Function segment_to_descriptor (seg : Word) : Word;

Description: Maps a real mode segment (paragraph) address onto an descriptor that can be used by a protected mode program to access the same memory. Parameters:

seg: the real mode segment you want the descriptor to.

Return values: Descriptor to real mode segment address. Notes: The returned descriptors limit will be set to 64 kB. Multiple calls to this function with the same segment address will return the same selector. Descriptors created by this function can never be modified or freed. Programs which need to examine various real mode addresses using the same selector should use the function allocate_ldt_descriptors ([85](#)) and change the base address as necessary.

Errors: Check the int31error variable.

See also: allocate_ldt_descriptors ([85](#)), free_ldt_descriptor ([91](#)), set_segment_base_address ([108](#))

For an example, see seg_fillchar ([104](#)).

seg_move

Declaration: `Procedure seg_move (sseg : Word; source : Longint; dseg : Word; dest : Longint; count : Longint);`

Description: Copies data between two memory locations. Parameters:

sseg: source selector.

source: source offset.

dseg: destination selector.

dest: destination offset.

count: size in bytes to copy.

Return values: None. Notes: Overlapping is only checked if the source selector is equal to the destination selector. No range check is done.

Errors: None.

See also: [seg_fillchar \(104\)](#), [seg_fillword \(105\)](#), [dosmemfillchar \(88\)](#), [dosmemfillword \(89\)](#), [dosmemget \(89\)](#), [dosmempu \(90\)](#), [dosmemmove \(90\)](#)

For an example, see [allocate_ldt_descriptors \(85\)](#).

set_descriptor_access_rights

Declaration: `Function set_descriptor_access_rights (d : Word; w : Word) : Longint;`

Description: Sets the access rights of a descriptor. Parameters:

d: selector.

w: new descriptor access rights.

Return values: This function doesn't return anything useful.

Errors: Check the `int31error` variable.

See also: [get_descriptor_access_rights \(92\)](#)

set_pm_interrupt

Declaration: `Function set_pm_interrupt (vector : byte; const intaddr : tseginfo) : boolean;`

Description: Sets the address of the protected mode handler for an interrupt. Parameters:

vector: number of protected mode interrupt to set.

intaddr: selector:offset address to the interrupt vector.

Return values: `True` if successful, `False` otherwise. Notes: The address supplied must be a valid `selector:offset` protected mode address.

Errors: Check the `int31error` variable.

See also: [get_pm_interrupt \(94\)](#), [set_rm_interrupt \(107\)](#), [get_rm_interrupt \(97\)](#)

```
uses
    crt,
    go32;

const
    int1c = $1c;

var
    oldint1c : tseginfo;
    newint1c : tseginfo;

    int1c_counter : Longint;

    int1c_ds : Word; external name '___v2prt0_ds_alias';

procedure int1c_handler; assembler;
asm
    cli
    pushw %ds
    pushw %ax
    movw %cs:int1c_ds, %ax
    movw %ax, %ds
    incl int1c_counter
    popw %ax
    popw %ds
    sti
    iret
end;

var i : Longint;

begin
    newint1c.offset := @int1c_handler;
    newint1c.segment := get_cs;
    get_pm_interrupt(int1c, oldint1c);
    WriteLn('-- Press any key to exit --');
    set_pm_interrupt(int1c, newint1c);
    while (not keypressed) do begin
        gotoxy(1, wherey);
        write('Number of interrupts occurred : ', int1c_counter);
    end;
    set_pm_interrupt(int1c, oldint1c);
end.
```

set_rm_interrupt

Declaration: Function set_rm_interrupt (vector : byte; const intaddr : tseginfo)
: boolean;

Description: Sets a real mode interrupt handler. Parameters:

vector: the interrupt vector number to set.

intaddr: address of new interrupt vector.

Return values: True if successful, otherwise False. **Notes:** The address supplied **MUST** be a real mode segment address, not a selector:offset address. So the interrupt handler must either

reside in DOS memory (below 1 Mb boundary) or the application must allocate a real mode callback address with `get_rm_callback` (95).

Errors: Check the `int31error` variable.

See also: `get_rm_interrupt` (97), `set_pm_interrupt` (106), `get_pm_interrupt` (94), `get_rm_callback` (95)

set_segment_base_address

Declaration: `Function set_segment_base_address (d : Word; s : Longint) : boolean;`

Description: Sets the 32-bit linear base address of a descriptor. Parameters:

d: selector.

s: new base address of the descriptor.

Errors: Check the `int31error` variable.

See also: `allocate_ldt_descriptors` (85), `get_segment_base_address` (98), `allocate_ldt_descriptors` (85), `set_segment_limit` (108), `get_segment_base_address` (98), `get_segment_limit` (99)

set_segment_limit

Declaration: `Function set_segment_limit (d : Word; s : Longint) : boolean;`

Description: Sets the limit of a descriptor. Parameters:

d: selector.

s: new limit of the descriptor.

Return values: Returns `True` if successful, else `False`. Notes: The new limit specified must be the byte length of the segment - 1. Segment limits bigger than or equal to 1MB must be page aligned, they must have the lower 12 bits set.

Errors: Check the `int31error` variable.

See also: `allocate_ldt_descriptors` (85), `set_segment_base_address` (108), `get_segment_limit` (99), `set_segment_limit` (108)

For an example, see `allocate_ldt_descriptors` (85).

tb_size

Declaration: `Function tb_size : Longint;`

Description: Returns the size of the pre-allocated DOS memory buffer. Parameters: None. Return values: The size of the pre-allocated DOS memory buffer. Notes: This block always seems to be 16k in size, but don't rely on this.

Errors: None.

See also: `transfer_buffer` (109), `copyfromdos` (87) `copytodos` (87)

transfer_buffer

Declaration: `Function transfer_buffer : Longint;`

Description: `transfer_buffer` returns the offset of the transfer buffer.

Errors: None.

See also: `tb_size` ([108](#))

unlock_code

Declaration: `Function unlock_code (functionaddr : pointer; size : Longint) : boolean;`

Description: Unlocks a memory range which resides in the code segment selector. Parameters:

functionaddr: address of function to be unlocked.

size: size bytes to be unlocked.

Return value: True if successful, False otherwise.

Errors: Check the `int31error` variable.

See also: `unlock_linear_region` ([109](#)), `unlock_data` ([109](#)), `lock_linear_region` ([102](#)), `lock_data` ([102](#)), `lock_code` ([102](#))

For an example, see `get_rm_callback` ([95](#)).

unlock_data

Declaration: `Function unlock_data (var data; size : Longint) : boolean;`

Description: Unlocks a memory range which resides in the data segment selector. Parameters:

data: address of memory to be unlocked.

size: size bytes to be unlocked.

Return values: True if successful, False otherwise.

Errors: Check the `int31error` variable.

See also: `unlock_linear_region` ([109](#)), `unlock_code` ([109](#)), `lock_linear_region` ([102](#)), `lock_data` ([102](#)), `lock_code` ([102](#))

For an example, see `get_rm_callback` ([95](#)).

unlock_linear_region

Declaration: `Function unlock_linear_region (linearaddr, size : Longint) : boolean;`

Description: Unlocks a previously locked linear region range to allow it to be swapped out again if needed.

Parameters:

linearaddr: linear address of the memory to be unlocked.

size: size bytes to be unlocked.

Return values: True if successful, False otherwise.

Errors: Check the `int31error` variable.

See also: `unlock_data` ([109](#)), `unlock_code` ([109](#)), `lock_linear_region` ([102](#)), `lock_data` ([102](#)), `lock_code` ([102](#))

Chapter 8

The GRAPH unit.

This document describes the GRAPH unit for Free Pascal, for all platforms. The unit was first written for DOS by Florian klämpfl, but was later completely rewritten by Carl-Eric Codere to be completely portable.

This chapter is divided in 4 sections.

- The first section gives an introduction to the graph unit.
- The second section lists the pre-defined constants, types and variables.
- The second section describes the functions which appear in the interface part of the GRAPH unit.
- The last part describes some system-specific issues.

8.1 Introduction

Requirements

The unit Graph exports functions and procedures for graphical output. It requires at least a VGA-compatible Card or a VGA-Card with software-driver (min. **512Kb** video memory).

A word about mode selection

The graph unit was implemented for compatibility with the old Turbo Pascal graph unit. For this reason, the mode constants as they were defined in the Turbo Pascal graph unit are retained.

However, since

1. Video cards have evolved very much
2. Free Pascal runs on multiple platforms

it was decided to implement new mode and graphic driver constants, which are more independent of the specific platform the program runs on.

In this section we give a short explanation of the new mode system. the following drivers were defined:


```
D1bit = 11;
D2bit = 12;
D4bit = 13;
D6bit = 14; { 64 colors Half-brite mode - Amiga }
D8bit = 15;
D12bit = 16; { 4096 color modes HAM mode - Amiga }
D15bit = 17;
D16bit = 18;
D24bit = 19; { not yet supported }
D32bit = 20; { not yet supported }
D64bit = 21; { not yet supported }

lowNewDriver = 11;
highNewDriver = 21;
```

Each of these drivers specifies a desired color-depth.

The following modes have been defined:

```
detectMode = 30000;
m320x200 = 30001;
m320x256 = 30002; { amiga resolution (PAL) }
m320x400 = 30003; { amiga/atari resolution }
m512x384 = 30004; { mac resolution }
m640x200 = 30005; { vga resolution }
m640x256 = 30006; { amiga resolution (PAL) }
m640x350 = 30007; { vga resolution }
m640x400 = 30008;
m640x480 = 30009;
m800x600 = 30010;
m832x624 = 30011; { mac resolution }
m1024x768 = 30012;
m1280x1024 = 30013;
m1600x1200 = 30014;
m2048x1536 = 30015;

lowNewMode = 30001;
highNewMode = 30015;
```

These modes start at 30000 because Borland specified that the mode number should be ascending with increasing X resolution, and the new constants shouldn't interfere with the old ones.

The above constants can be used to set a certain color depth and resolution, as demonstrated in the following example:

Listing: graphex/inigraph1.pp

Program inigraph1;

```
{ Program to demonstrate static graphics mode selection }
```

```
uses graph;
```

```
const
```

```
  TheLine = 'We are now in 640 x 480 x 256 colors!'+
    ' (press <Return> to continue)';
```

```
var
  gd, gm, lo, hi, error, tw, th: integer;
  found: boolean;

begin
  { We want an 8 bit mode }
  gd := D8bit;
  gm := m640x480;
  initgraph(gd, gm, '');
  { Make sure you always check graphresult! }
  error := graphResult;
  if (error <> grOk) Then
    begin
      writeln('640x480x256 is not supported!');
      halt(1)
    end;
  { We are now in 640x480x256 }
  setColor(cyan);
  rectangle(0,0,getmaxx,getmaxy);
  { Write a nice message in the center of the screen }
  setTextStyle(defaultFont, horizDir, 1);
  tw:=TextWidth(TheLine);
  th:=TextHeight(TheLine);
  outTextXY((getMaxX - TW) div 2,
            (getMaxY - TH) div 2, TheLine);
  { Wait for return }
  readln;
  { Back to text mode }
  closegraph;
end.
```

If other modes than the ones above are supported by the graphics card, you will not be able to select them with this mechanism.

For this reason, there is also a 'dynamic' mode number, which is assigned at run-time. This number increases with increasing X resolution. It can be queried with the `getmoderange` call. This call will return the range of modes which are valid for a certain graphics driver. The numbers are guaranteed to be consecutive, and can be used to search for a certain resolution, as in the following example:

Listing: graphex/inigraph2.pp

```
Program inigraph2;

{ Program to demonstrate dynamic graphics mode selection }

uses graph;

const
  TheLine = 'We are now in 640 x 480 x 256 colors!'+
            ' (press <Return> to continue)';

var
  th, tw, gd, gm, lo, hi, error: integer;
  found: boolean;

begin
  { We want an 8 bit mode }
  gd := D8bit;
```

```
{ Get all available resolutions for this bitdepth }
getmoderange(gd,lo,hi);
{ If the highest available mode number is -1,
  no resolutions are supported for this bitdepth }
if hi = -1 then
  begin
    writeln('no 8 bit modes supported!');
    halt
  end;
found := false;
{ Search all resolutions for 640x480 }
for gm := lo to hi do
  begin
    initgraph(gd,gm,'');
    { Make sure you always check graphresult! }
    error := graphResult;
    if (error = grOk) and
      (getmaxx = 639) and (getmaxy = 479) then
      begin
        found := true;
        break;
      end;
  end;
if not found then
  begin
    writeln('640x480x256 is not supported!');
    halt(1)
  end;
{ We are now in 640x480x256 }
setColor(cyan);
rectangle(0,0,getmaxx,getmaxy);
{ Write a nice message in the center of the screen }
setTextStyle(defaultFont,horizDir,1);
TW:=TextWidth(TheLine);
TH:=TextHeight(TheLine);
outTextXY((getMaxX - TW) div 2,
          (getMaxY - TH) div 2,TheLine);
{ Wait for return }
readln;
{ Back to text mode }
closegraph;
end.
```

Thus, the getmoderange function can be used to detect all available modes and drivers, as in the following example:

Listing: graphex/modrange.pp

Program GetModeRange_Example;

```
{ This program demonstrates how to find all available graph modes }

uses graph;

const
  { Currently, only 4, 8, 15 and 16 bit modes are supported
    but this may change in the future }
  gdnames: array[D4bit..D16bit] of string[6] =
    ('4 bit','6 bit','8 bit','12 bit','15 bit','16 bit');
```

```
var
  t: text;
  line : string;
  gd, c, low, high, res: integer;
begin
  assign(t, 'modes.txt');
  rewrite(t);
  close(t);
  for gd := D4bit to D16bit do
    begin
      { Get the available mode numbers for this driver }
      getModeRange(gd,low,high);
      append(t);
      write(t,gdnames[gd]);
      Writeln(t,': low modenr = ',low,', high modenr = ',high);
      close(t);
      { If high is -1,
        no resolutions are supported for this bitdepth }
      if high = -1 then
        begin
          append(t);
          writeln(t, ' No modes supported!');
          writeln(t);
          close(t);
        end
      else
        { Enter all supported resolutions for this bitdepth
          and write their characteristics to the file }
        for c := low to high do
          begin
            append(t);
            writeln(t,' testing mode nr ',c);
            close(t);
            initgraph(gd,c,'');
            res := graphresult;
            append(t);
            { An error occurred when entering the mode? }
            if res <> grok then
              writeln(t,grapherrormsg(res))
            else
              begin
                write(t,'maxx: ',getmaxx,', maxy: ',getmaxy);
                Writeln(t,', maxcolor: ',getmaxcolor);
                closegraph;
              end;
              writeln(t);
              close(t);
            end;
            append(t);
            writeln(t);
            close(t);
          end;
        Writeln('All supported modes are listed in modes.txt files');
      end.
```

8.2 Constants, Types and Variables

Types

```
ArcCoordsType = record
  X,Y,Xstart,Ystart,Xend,Yend : Integer;
end;
FillPatternType = Array [1..8] of Byte;
FillSettingsType = Record
  Pattern,Color : Word
end;
LineSettingsType = Record
  LineStyle,Pattern, Width : Word;
end;
PaletteType = Record
  Size : Byte;
  Colors : array[0..MAXcolor] of shortint;
end;
PointType = Record
  X,Y : Integer;
end;
TextSettingsType = Record
  Font,Direction, CharSize, Horiz, Vert : Word
end;
ViewPortType = Record
  X1,Y1,X2,Y2 : Integer;
  Clip : Boolean
end;
```

8.3 Function list by category

What follows is a listing of the available functions, grouped by category. For each function there is a reference to the page where you can find the function.

Initialization

Initialization of the graphics screen.

Name	Description	Page
ClearDevice	Empty the graphics screen	119
CloseGraph	Finish drawing session, return to text mode	119
DetectGraph	Detect graphical modes	119
GetAspectRatio	Get aspect ratio of screen	121
GetModeRange	Get range of valid modes for current driver	123
GraphDefaults	Set defaults	125
GetDriverName	Return name of graphical driver	121
GetGraphMode	Return current or last used graphics mode	122
GetMaxMode	Get maximum mode for current driver	123

GetModeName	Get name of current mode	123
GraphErrorMsg	String representation of graphical error	125
GraphResult	Result of last drawing operation	125
InitGraph	Initialize graphics drivers	126
InstallUserDriver	Install a new driver	126
RegisterBGIDriver	Register a new driver	129
RestoreCRTMode	Go back to text mode	129
SetGraphBufSize	Set buffer size for graphical operations	131
SetGraphMode	Set graphical mode	132

screen management

General drawing screen management functions.

Name	Description	Page
ClearViewPort	Clear the current viewport	119
GetImage	Copy image from screen to memory	122
GetMaxX	Get maximum X coordinate	123
GetMaxY	Get maximum Y coordinate	123
GetX	Get current X position	124
GetY	Get current Y position	125
ImageSize	Get size of selected image	126
GetViewSettings	Get current viewport settings	124
PutImage	Copy image from memory to screen	128
SetActivePage	Set active video page	130
SetAspectRatio	Set aspect ratio for drawing routines	130
SetViewPort	Set current viewport	134
SetVisualPage	Set visual page	134
SetWriteMode	Set write mode for screen operations	134

Color management

All functions related to color management.

Name	Description	Page
GetBkColor	Get current background color	121
GetColor	Get current foreground color	121
GetDefaultPalette	Get default palette entries	121
GetMaxColor	Get maximum valid color	122
GetPaletteSize	Get size of palette for current mode	124
GetPixel	Get color of selected pixel	124
GetPalette	Get palette entry	124

SetAllPalette	Set all colors in palette	130
SetBkColor	Set background color	130
SetColor	Set foreground color	131
SetPalette	Set palette entry	132
SetRGBPalette	Set palette entry with RGB values	132

Drawing primitives

Functions for simple drawing.

Name	Description	Page
Arc	Draw an arc	118
Circle	Draw a complete circle	119
DrawPoly	Draw a polygon with N points	120
Ellipse	Draw an ellipse	120
GetArcCoords	Get arc coordinates	121
GetLineSettings	Get current line drawing settings	122
Line	Draw line between 2 points	127
LineRel	Draw line relative to current position	127
LineTo	Draw line from current position to absolute position	127
MoveRel	Move cursor relative to current position	127
MoveTo	Move cursor to absolute position	128
PieSlice	Draw a pie slice	128
PutPixel	Draw 1 pixel	129
Rectangle	Draw a non-filled rectangle	129
Sector	Draw a sector	130
SetLineStyle	Set current line drawing style	132

Filled drawings

Functions for drawing filled regions.

Name	Description	Page
Bar3D	Draw a filled 3D-style bar	118
Bar	Draw a filled rectangle	118
FloodFill	Fill starting from coordinate	120
FillEllipse	Draw a filled ellipse	120
FillPoly	Draw a filled polygon	120
GetFillPattern	Get current fill pattern	122
GetFillSettings	Get current fill settings	122
SetFillPattern	Set current fill pattern	131
SetFillStyle	Set current fill settings	131

Text and font handling

Functions to set texts on the screen.

Name	Description	Page
GetTextSettings	Get current text settings	124
InstallUserFont	Install a new font	127
OutText	Write text at current cursor position	128
OutTextXY	Write text at coordinates X,Y	128
RegisterBGIFont	Register a new font	129
SetTextJustify	Set text justification	133
SetTextStyle	Set text style	133
SetUserCharSize	Set text size	133
TextHeight	Calculate height of text	134
TextWidth	Calculate width of text	134

8.4 Functions and procedures

Arc

Declaration: `Procedure Arc (X,Y : Integer; start,stop, radius : Word);`

Description: Arc draws part of a circle with center at (X,Y), radius radius, starting from angle start, stopping at angle stop. These angles are measured counterclockwise.

Errors: None.

See also: Circle ([119](#)), Ellipse ([120](#)) GetArcCoords ([121](#)), PieSlice ([128](#)), Sector ([130](#))

Bar

Declaration: `Procedure Bar (X1,Y1,X2,Y2 : Integer);`

Description: Draws a rectangle with corners at (X1,Y1) and (X2,Y2) and fills it with the current color and fill-style.

Errors: None.

See also: Bar3D ([118](#)), Rectangle ([129](#))

Bar3D

Declaration: `Procedure Bar3D (X1,Y1,X2,Y2 : Integer; depth : Word; Top : Boolean);`

Description: Draws a 3-dimensional Bar with corners at (X1,Y1) and (X2,Y2) and fills it with the current color and fill-style. Depth specifies the number of pixels used to show the depth of the bar. If Top is true; then a 3-dimensional top is drawn.

Errors: None.

See also: Bar ([118](#)), Rectangle ([129](#))

Circle

Declaration: `Procedure Circle (X,Y : Integer; Radius : Word);`

Description: `Circle` draws part of a circle with center at `(X,Y)`, radius `radius`.

Errors: None.

See also: `Ellipse` ([120](#)), `Arc` ([118](#)) `GetArcCoords` ([121](#)), `PieSlice` ([128](#)), `Sector` ([130](#))

ClearDevice

Declaration: `Procedure ClearDevice ;`

Description: Clears the graphical screen (with the current background color), and sets the pointer at `(0 , 0)`

Errors: None.

See also: `ClearViewPort` ([119](#)), `SetBkColor` ([130](#))

ClearViewPort

Declaration: `Procedure ClearViewPort ;`

Description: Clears the current viewport. The current background color is used as filling color. The pointer is set at `(0 , 0)`

Errors: None.

See also: `ClearDevice` ([119](#)), `SetViewPort` ([134](#)), `SetBkColor` ([130](#))

CloseGraph

Declaration: `Procedure CloseGraph ;`

Description: Closes the graphical system, and restores the screen modus which was active before the graphical modus was activated.

Errors: None.

See also: `InitGraph` ([126](#))

DetectGraph

Declaration: `Procedure DetectGraph (Var Driver, Modus : Integer);`

Description: Checks the hardware in the PC and determines the driver and screen-modus to be used. These are returned in `Driver` and `Modus`, and can be fed to `InitGraph`. See the `InitGraph` for a list of drivers and modi.

Errors: None.

See also: `InitGraph` ([126](#))

DrawPoly

Declaration: `Procedure DrawPoly (NumberOfPoints : Word; Var PolyPoints;`

Description: Draws a polygone with `NumberOfPoints` corner points, using the current color and line-style. `PolyPoints` is an array of type `PointType`.

Errors: None.

See also: [Bar \(118\)](#), [seepBar3D](#), [Rectangle \(129\)](#)

Ellipse

Declaration: `Procedure Ellipse (X,Y : Integer; Start,Stop,XRadius,YRadius : Word);`

Description: `Ellipse` draws part of an ellipse with center at `(X,Y)`. `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. `Start` and `Stop` are the starting and stopping angles of the part of the ellipse. They are measured counterclockwise from the X-axis (3 o'clock is equal to 0 degrees). Only positive angles can be specified.

Errors: None.

See also: [Arc \(118\)](#) [Circle \(119\)](#), [FillEllipse \(120\)](#)

FillEllipse

Declaration: `Procedure FillEllipse (X,Y : Integer; Xradius,YRadius: Word);`

Description: `Ellipse` draws an ellipse with center at `(X,Y)`. `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. The ellipse is filled with the current color and fill-style.

Errors: None.

See also: [Arc \(118\)](#) [Circle \(119\)](#), [GetArcCoords \(121\)](#), [PieSlice \(128\)](#), [Sector \(130\)](#)

FillPoly

Declaration: `Procedure FillPoly (NumberOfPoints : Word; Var PolyPoints);`

Description: Draws a polygone with `NumberOfPoints` corner points and fills it using the current color and line-style. `PolyPoints` is an array of type `PointType`.

Errors: None.

See also: [Bar \(118\)](#), [seepBar3D](#), [Rectangle \(129\)](#)

FloodFill

Declaration: `Procedure FloodFill (X,Y : Integer; BorderColor : Word);`

Description: Fills the area containing the point `(X,Y)`, bounded by the color `BorderColor`.

Errors: None

See also: [SetColor \(131\)](#), [SetBkColor \(130\)](#)

GetArcCoords

Declaration: `Procedure GetArcCoords (Var ArcCoords : ArcCoordsType);`

Description: `GetArcCoords` returns the coordinates of the latest `Arc` or `Ellipse` call.

Errors: None.

See also: `Arc` ([118](#)), `Ellipse` ([120](#))

GetAspectRatio

Declaration: `Procedure GetAspectRatio (Var Xasp, Yasp : Word);`

Description: `GetAspectRatio` determines the effective resolution of the screen. The aspect ration can the be calculated as `Xasp/Yasp`.

Errors: None.

See also: `InitGraph` ([126](#)), `SetAspectRatio` ([130](#))

GetBkColor

Declaration: `Function GetBkColor : Word;`

Description: `GetBkColor` returns the current background color (the palette entry).

Errors: None.

See also: `GetColor` ([121](#)), `SetBkColor` ([130](#))

GetColor

Declaration: `Function GetColor : Word;`

Description: `GetColor` returns the current drawing color (the palette entry).

Errors: None.

See also: `GetColor` ([121](#)), `SetBkColor` ([130](#))

GetDefaultPalette

Declaration: `Procedure GetDefaultPalette (Var Palette : PaletteType);`

Description: Returns the current palette in `Palette`.

Errors: None.

See also: `GetColor` ([121](#)), `GetBkColor` ([121](#))

GetDriverName

Declaration: `Function GetDriverName : String;`

Description: `GetDriverName` returns a string containing the name of the current driver.

Errors: None.

See also: `GetModeName` ([123](#)), `InitGraph` ([126](#))

GetFillPattern

Declaration: `Procedure GetFillPattern (Var FillPattern : FillPatternType);`

Description: `GetFillPattern` returns an array with the current fill-pattern in `FillPattern`

Errors: None

See also: `SetFillPattern` ([131](#))

GetFillSettings

Declaration: `Procedure GetFillSettings (Var FillInfo : FillSettingsType);`

Description: `GetFillSettings` returns the current fill-settings in `FillInfo`

Errors: None.

See also: `SetFillPattern` ([131](#))

GetGraphMode

Declaration: `Function GetGraphMode : Integer;`

Description: `GetGraphMode` returns the current graphical modulus

Errors: None.

See also: `InitGraph` ([126](#))

GetImage

Declaration: `Procedure GetImage (X1,Y1,X2,Y2 : Integer, Var Bitmap;`

Description: `GetImage` Places a copy of the screen area (X1,Y1) to X2,Y2 in `BitMap`

Errors: `Bitmap` must have enough room to contain the image.

See also: `ImageSize` ([126](#)), `PutImage` ([128](#))

GetLineSettings

Declaration: `Procedure GetLineSettings (Var LineInfo : LineSettingsType);`

Description: `GetLineSettings` returns the current Line settings in `LineInfo`

Errors: None.

See also: `SetLineStyle` ([132](#))

GetMaxColor

Declaration: `Function GetMaxColor : Word;`

Description: `GetMaxColor` returns the maximum color-number which can be set with `SetColor`. Contrary to Turbo Pascal, this color isn't always guaranteed to be white (for instance in 256+ color modes).

Errors: None.

See also: `SetColor` ([131](#)), `GetPaletteSize` ([124](#))

GetMaxMode

Declaration: `Function GetMaxMode : Word;`

Description: `GetMaxMode` returns the highest modus for the current driver.

Errors: None.

See also: `InitGraph` ([126](#))

GetMaxX

Declaration: `Function GetMaxX : Word;`

Description: `GetMaxX` returns the maximum horizontal screen length

Errors: None.

See also: `GetMaxY` ([123](#))

GetMaxY

Declaration: `Function GetMaxY : Word;`

Description: `GetMaxY` returns the maximum number of screen lines

Errors: None.

See also: `GetMaxY` ([123](#))

GetModeName

Declaration: `Function GetModeName (Var modus : Integer) : String;`

Description: Returns a string with the name of modus `Modus`

Errors: None.

See also: `GetDriverName` ([121](#)), `InitGraph` ([126](#))

GetModeRange

Declaration: `Procedure GetModeRange (Driver : Integer;
LoModus, HiModus: Integer);`

Description: `GetModeRange` returns the Lowest and Highest modus of the currently installed driver. If no modes are supported for this driver, `HiModus` will be -1.

Errors: None.

See also: `InitGraph` ([126](#))

GetPalette

Declaration: `Procedure GetPalette (Var Palette : PaletteType);`

Description: `GetPalette` returns in `Palette` the current palette.

Errors: None.

See also: `GetPaletteSize` ([124](#)), `SetPalette` ([132](#))

GetPaletteSize

Declaration: `Function GetPaletteSize : Word;`

Description: `GetPaletteSize` returns the maximum number of entries in the current palette.

Errors: None.

See also: `GetPalette` ([124](#)), `SetPalette` ([132](#))

GetPixel

Declaration: `Function GetPixel (X,Y : Integer) : Word;`

Description: `GetPixel` returns the color of the point at `(X,Y)`

Errors: None.

See also:

GetTextSettings

Declaration: `Procedure GetTextSettings (Var TextInfo : TextSettingsType);`

Description: `GetTextSettings` returns the current text style settings : The font, direction, size and placement as set with `SetTextStyle` and `SetTextJustify`

Errors: None.

See also: `SetTextStyle` ([133](#)), `SetTextJustify` ([133](#))

GetViewSettings

Declaration: `Procedure GetViewSettings (Var ViewPort : ViewPortType);`

Description: `GetViewSettings` returns the current viewport and clipping settings in `ViewPort`.

Errors: None.

See also: `SetViewPort` ([134](#))

GetX

Declaration: `Function GetX : Integer;`

Description: `GetX` returns the X-coordinate of the current position of the graphical pointer

Errors: None.

See also: `GetY` ([125](#))

GetY

Declaration: `Function GetY : Integer;`

Description: `GetY` returns the Y-coordinate of the current position of the graphical pointer

Errors: None.

See also: `GetX` ([124](#))

GraphDefaults

Declaration: `Procedure GraphDefaults ;`

Description: `GraphDefaults` resets all settings for viewport, palette, foreground and background pattern, line-style and pattern, filling style, filling color and pattern, font, text-placement and text size.

Errors: None.

See also: `SetViewPort` ([134](#)), `SetFillStyle` ([131](#)), `SetColor` ([131](#)), `SetBkColor` ([130](#)), `SetLineStyle` ([132](#))

GraphErrorMsg

Declaration: `Function GraphErrorMsg (ErrorCode : Integer) : String;`

Description: `GraphErrorMsg` returns a string describing the error `Errorcode`. This string can be used to let the user know what went wrong.

Errors: None.

See also: `GraphResult` ([125](#))

GraphResult

Declaration: `Function GraphResult : Integer;`

Description: `GraphResult` returns an error-code for the last graphical operation. If the returned value is zero, all went well. A value different from zero means an error has occurred. besides all operations which draw something on the screen, the following procedures also can produce a `GraphResult` different from zero:

- `InstallUserFont` ([127](#))
- `SetLineStyle` ([132](#))
- `SetWriteMode` ([134](#))
- `SetFillStyle` ([131](#))
- `SetTextJustify` ([133](#))
- `SetGraphMode` ([132](#))
- `SetTextStyle` ([133](#))

Errors: None.

See also: `GraphErrorMsg` ([125](#))

ImageSize

Declaration: `Function ImageSize (X1,Y1,X2,Y2 : Integer) : Word;`

Description: `ImageSize` returns the number of bytes needed to store the image in the rectangle defined by (X1,Y1) and (X2,Y2).

Errors: None.

See also: `GetImage` ([122](#))

InitGraph

Declaration: `Procedure InitGraph (var GraphDriver,GraphModus : integer;
const PathToDriver : string);`

Description: `InitGraph` initializes the graph package. `GraphDriver` has two valid values: `GraphDriver=0` which performs an auto detect and initializes the highest possible mode with the most colors. 1024x768x64K is the highest possible resolution supported by the driver, if you need a higher resolution, you must edit `MODES.PPI`. If you need another mode, then set `GraphDriver` to a value different from zero and `graphmode` to the mode you wish (VESA modes where 640x480x256 is 101h etc.). `PathToDriver` is only needed, if you use the BGI fonts from Borland.

Errors: None.

See also: Introduction, (page [110](#)), `DetectGraph` ([119](#)), `CloseGraph` ([119](#)), `GraphResult` ([125](#))

Example:

```
var
    gd,gm : integer;
    PathToDriver : string;
begin
    gd:=detect; { highest possible resolution }
    gm:=0; { not needed, auto detection }
    PathToDriver:='C:\PP\BGI'; { path to BGI fonts,
                                drivers aren't needed }

    InitGraph(gd,gm,PathToDriver);
    if GraphResult<>grok then
        halt; ..... { whatever you need }
    CloseGraph; { restores the old graphics mode }
end.
```

InstallUserDriver

Declaration: `Function InstallUserDriver (DriverPath : String;
AutoDetectPtr: Pointer) : Integer;`

Description: `InstallUserDriver` adds the device-driver `DriverPath` to the list of .BGI drivers. `AutoDetectPtr` is a pointer to a possible auto-detect function.

Errors: None.

See also: `InitGraph` ([126](#)), `InstallUserFont` ([127](#))

InstallUserFont

Declaration: `Function InstallUserFont (FontPath : String) : Integer;`

Description: `InstallUserFont` adds the font in `FontPath` to the list of fonts of the .BGI system.

Errors: None.

See also: `InitGraph` ([126](#)), `InstallUserDriver` ([126](#))

Line

Declaration: `Procedure Line (X1,Y1,X2,Y2 : Integer);`

Description: `Line` draws a line starting from `(X1,Y1)` to `(X2,Y2)`, in the current line style and color. The current position is put to `(X2,Y2)`

Errors: None.

See also: `LineRel` ([127](#)), `LineTo` ([127](#))

LineRel

Declaration: `Procedure LineRel (DX,DY : Integer);`

Description: `LineRel` draws a line starting from the current pointer position to the point `(DX,DY)`, **relative** to the current position, in the current line style and color. The Current Position is set to the endpoint of the line.

Errors: None.

See also: `Line` ([127](#)), `LineTo` ([127](#))

LineTo

Declaration: `Procedure LineTo (DX,DY : Integer);`

Description: `LineTo` draws a line starting from the current pointer position to the point `(DX,DY)`, **relative** to the current position, in the current line style and color. The Current position is set to the end of the line.

Errors: None.

See also: `LineRel` ([127](#)), `Line` ([127](#))

MoveRel

Declaration: `Procedure MoveRel (DX,DY : Integer);`

Description: `MoveRel` moves the pointer to the point `(DX,DY)`, relative to the current pointer position

Errors: None.

See also: `MoveTo` ([128](#))

MoveTo

Declaration: `Procedure MoveTo (X,Y : Integer;`

Description: `MoveTo` moves the pointer to the point (X,Y).

Errors: None.

See also: `MoveRel` ([127](#))

OutText

Declaration: `Procedure OutText (Const TextString : String);`

Description: `OutText` puts `TextString` on the screen, at the current pointer position, using the current font and text settings. The current position is moved to the end of the text.

Errors: None.

See also: `OutTextXY` ([128](#))

OutTextXY

Declaration: `Procedure OutTextXY (X,Y : Integer; Const TextString : String);`

Description: `OutText` puts `TextString` on the screen, at position (X,Y), using the current font and text settings. The current position is moved to the end of the text.

Errors: None.

See also: `OutText` ([128](#))

PieSlice

Declaration: `Procedure PieSlice (X,Y : Integer;
Start,Stop,Radius : Word);`

Description: `PieSlice` draws and fills a sector of a circle with center (X,Y) and radius `Radius`, starting at angle `Start` and ending at angle `Stop`.

Errors: None.

See also: `Arc` ([118](#)), `Circle` ([119](#)), `Sector` ([130](#))

PutImage

Declaration: `Procedure PutImage (X1,Y1 : Integer; Var Bitmap; How : word) ;`

Description: `PutImage` Places the bitmap in `Bitmap` on the screen at (X1,Y1). `How` determines how the bitmap will be placed on the screen. Possible values are :

- `CopyPut`
- `XORPut`
- `ORPut`
- `AndPut`
- `NotPut`

Errors: None

See also: [ImageSize \(126\)](#), [GetImage \(122\)](#)

PutPixel

Declaration: `Procedure PutPixel (X,Y : Integer; Color : Word);`

Description: Puts a point at (X,Y) using color Color

Errors: None.

See also: [GetPixel \(124\)](#)

Rectangle

Declaration: `Procedure Rectangle (X1,Y1,X2,Y2 : Integer);`

Description: Draws a rectangle with corners at (X1,Y1) and (X2,Y2), using the current color and style.

Errors: None.

See also: [Bar \(118\)](#), [Bar3D \(118\)](#)

RegisterBGIDriver

Declaration: `Function RegisterBGIDriver (Driver : Pointer) : Integer;`

Description: Registers a user-defined BGI driver

Errors: None.

See also: [InstallUserDriver \(126\)](#), [RegisterBGIFont \(129\)](#)

RegisterBGIFont

Declaration: `Function RegisterBGIFont (Font : Pointer) : Integer;`

Description: Registers a user-defined BGI driver

Errors: None.

See also: [InstallUserFont \(127\)](#), [RegisterBGIDriver \(129\)](#)

RestoreCRTMode

Declaration: `Procedure RestoreCRTMode ;`

Description: Restores the screen modus which was active before the graphical modus was started.

To get back to the graph mode you were last in, you can use `SetGraphMode (GetGraphMode)`

Errors: None.

See also: [InitGraph \(126\)](#)

Sector

Declaration: `Procedure Sector (X,Y : Integer;
Start,Stop,XRadius,YRadius : Word);`

Description: `Sector` draws and fills a sector of an ellipse with center (X,Y) and radii `XRadius` and `YRadius`, starting at angle `Start` and ending at angle `Stop`.

Errors: None.

See also: [Arc \(118\)](#), [Circle \(119\)](#), [PieSlice \(128\)](#)

SetActivePage

Declaration: `Procedure SetActivePage (Page : Word);`

Description: Sets `Page` as the active page for all graphical output.

Errors: None.

See also:

SetAllPalette

Declaration: `Procedure SetAllPalette (Var Palette);`

Description: Sets the current palette to `Palette`. `Palette` is an untyped variable, usually pointing to a record of type `PaletteType`

Errors: None.

See also: [GetPalette \(124\)](#)

SetAspectRatio

Declaration: `Procedure SetAspectRatio (Xasp,Yasp : Word);`

Description: Sets the aspect ratio of the current screen to `Xasp/Yasp`.

Errors: None

See also: [InitGraph \(126\)](#), [GetAspectRatio \(121\)](#)

SetBkColor

Declaration: `Procedure SetBkColor (Color : Word);`

Description: Sets the background color to `Color`.

Errors: None.

See also: [GetBkColor \(121\)](#), [SetColor \(131\)](#)

SetColor

Declaration: `Procedure SetColor (Color : Word);`

Description: Sets the foreground color to Color.

Errors: None.

See also: [GetColor \(121\)](#), [SetBkColor \(130\)](#)

SetFillPattern

Declaration: `Procedure SetFillPattern (FillPattern : FillPatternType,
Color : Word);`

Description: `SetFillPattern` sets the current fill-pattern to `FillPattern`, and the filling color to `Color`.
The pattern is an 8x8 raster, corresponding to the 64 bits in `FillPattern`.

Errors: None

See also: [GetFillPattern \(122\)](#), [SetFillStyle \(131\)](#)

SetFillStyle

Declaration: `Procedure SetFillStyle (Pattern,Color : word);`

Description: `SetFillStyle` sets the filling pattern and color to one of the predefined filling patterns. `Pattern` can be one of the following predefined constants :

- `EmptyFill` Uses backgroundcolor.
- `SolidFill` Uses filling color
- `LineFill` Fills with horizontal lines.
- `ltSlashFill` Fills with lines from left-under to top-right.
- `SlashFill` Idem as previous, thick lines.
- `BkSlashFill` Fills with thick lines from left-Top to bottom-right.
- `LtBkSlashFill` Idem as previous, normal lines.
- `HatchFill` Fills with a hatch-like pattern.
- `XHatchFill` Fills with a hatch pattern, rotated 45 degrees.
- `InterLeaveFill`
- `WideDotFill` Fills with dots, wide spacing.
- `CloseDotFill` Fills with dots, narrow spacing.
- `UserFill` Fills with a user-defined pattern.

Errors: None.

See also: [SetFillPattern \(131\)](#)

SetGraphBufSize

Declaration: `Procedure SetGraphBufSize (BufSize : Word);`

Description: `SetGraphBufSize` is a dummy function which does not do anything; it is no longer needed.

Errors: None.

See also:

SetGraphMode

Declaration: `Procedure SetGraphMode (Mode : Integer);`

Description: `SetGraphMode` sets the graphical mode and clears the screen.

Errors: None.

See also: `InitGraph` ([126](#))

SetLineStyle

Declaration: `Procedure SetLineStyle (LineStyle, Pattern, Width : Word);`

Description: `SetLineStyle` sets the drawing style for lines. You can specify a `LineStyle` which is one of the following pre-defined constants:

- `SolidLn=0`; draws a solid line.
- `DottedLn=1`; Draws a dotted line.
- `CenterLn=2`; draws a non-broken centered line.
- `DashedLn=3`; draws a dashed line.
- `UserBitLn=4`; Draws a User-defined bit pattern.

If `UserBitLn` is specified then `Pattern` contains the bit pattern. In all another cases, `Pattern` is ignored. The parameter `Width` indicates how thick the line should be. You can specify one of the following pre-defined constants:

- `NormWidth=1`
- `ThickWidth=3`

Errors: None.

See also: `GetLineSettings` ([122](#))

SetPalette

Declaration: `Procedure SetPalette (ColorNr : Word; NewColor : ShortInt);`

Description: `SetPalette` changes the `ColorNr`-th entry in the palette to `NewColor`

Errors: None.

See also: `SetAllPalette` ([130](#)), `SetRGBPalette` ([132](#))

SetRGBPalette

Declaration: `Procedure SetRGBPalette (ColorNr, Red, Green, Blue : Integer);`

Description: `SetRGBPalette` sets the `ColorNr`-th entry in the palette to the color with RGB-values `Red`, `Green` `Blue`.

Errors: None.

See also: `SetAllPalette` ([130](#)), `SetPalette` ([132](#))

SetTextJustify

Declaration: `Procedure SetTextJustify (Horizontal,Vertical : Word);`

Description: `SetTextJustify` controls the placement of new text, relative to the (graphical) cursor position. `Horizontal` controls horizontal placement, and can be one of the following pre-defined constants:

- `LeftText=0`; Text is set left of the pointer.
- `CenterText=1`; Text is set centered horizontally on the pointer.
- `RightText=2`; Text is set to the right of the pointer.

`Vertical` controls the vertical placement of the text, relative to the (graphical) cursor position. Its value can be one of the following pre-defined constants :

- `BottomText=0`; Text is placed under the pointer.
- `CenterText=1`; Text is placed centered vertically on the pointer.
- `TopText=2`; Text is placed above the pointer.

Errors: None.

See also: `OutText` ([128](#)), `OutTextXY` ([128](#))

SetTextStyle

Declaration: `Procedure SetTextStyle (Font,Direction,Magnitude : Word);`

Description: `SetTextStyle` controls the style of text to be put on the screen. pre-defined constants for `Font` are:

- `DefaultFont=0`;
- `TriplexFont=2`;
- `SmallFont=2`;
- `SansSerifFont=3`;
- `GothicFont=4`;

Pre-defined constants for `Direction` are :

- `HorizDir=0`;
- `VertDir=1`;

Errors: None.

See also: `GetTextSettings` ([124](#))

SetUserCharSize

Declaration: `Procedure SetUserCharSize (Xasp1,Xasp2,Yasp1,Yasp2 : Word);`

Description: Sets the width and height of vector-fonts. The horizontal size is given by `Xasp1/Xasp2`, and the vertical size by `Yasp1/Yasp2`.

Errors: None.

See also: `SetTextStyle` ([133](#))

SetViewPort

Declaration: `Procedure SetViewPort (X1,Y1,X2,Y2 : Integer; Clip : Boolean);`

Description: Sets the current graphical viewport (window) to the rectangle defined by the top-left corner (X1,Y1) and the bottom-right corner (X2,Y2). If Clip is true, anything drawn outside the viewport (window) will be clipped (i.e. not drawn). Coordinates specified after this call are relative to the top-left corner of the viewport.

Errors: None.

See also: [GetViewSettings \(124\)](#)

SetVisualPage

Declaration: `Procedure SetVisualPage (Page : Word);`

Description: `SetVisualPage` sets the video page to page number Page.

Errors: None

See also: [SetActivePage \(130\)](#)

SetWriteMode

Declaration: `Procedure SetWriteMode (Mode : Integer);`

Description: `SetWriteMode` controls the drawing of lines on the screen. It controls the binary operation used when drawing lines on the screen. Mode can be one of the following pre-defined constants:

- CopyPut=0;
- XORPut=1;

Errors: None.

See also:

TextHeight

Declaration: `Function TextHeight (S : String) : Word;`

Description: `TextHeight` returns the height (in pixels) of the string S in the current font and text-size.

Errors: None.

See also: [TextWidth \(134\)](#)

TextWidth

Declaration: `Function TextWidth (S : String) : Word;`

Description: `TextWidth` returns the width (in pixels) of the string S in the current font and text-size.

Errors: None.

See also: [TextHeight \(134\)](#)

8.5 Target specific issues

In what follows we describe some things that are different on the various platforms:

DOS

WINDOWS

LINUX

Chapter 9

The HEAPTRC unit.

This chapter describes the HEAPTRC unit for Free Pascal. It was written by Pierre Muller. It is system independent, and works on all supported systems.

9.1 Purpose

The HEAPTRC unit can be used to debug your memory allocation/deallocation. It keeps track of the calls to `getmem/freemem`, and, implicitly, of `New/Dispose` statements.

When the program exits, or when you request it explicitly. It displays the total memory used, and then dumps a list of blocks that were allocated but not freed. It also displays where the memory was allocated.

If there are any inconsistencies, such as memory blocks being allocated or freed twice, or a memory block that is released but with wrong size, this will be displayed also.

The information that is stored/displayed can be customized using some constants.

9.2 Usage

All that you need to do is to include `heaptrc` in the `uses` clause of your program. Make sure that it is the first unit in the clause, otherwise memory allocated in initialization code of units that precede the `heaptrc` unit will not be accounted for, causing an incorrect memory usage report.

If you use the `-gh` switch, the compiler will insert the unit by itself, so you don't have to include it in your `uses` clause.

The following example shows how to use the `heaptrc` unit.

```
Program heapex;  
  
{ Program used to demonstrate the usage of heaptrc unit }  
  
Uses heaptrc;  
  
Var P1 : ^Longint;  
      P2 : Pointer;  
      I : longint;  
  
begin  
  New(P1);
```

```
// causes previous allocation not to be de-allocated
New(P1);
Dispose(P1);
For I:=1 to 10 do
  begin
    GetMem (P2,128);
    // When I is even, deallocate block. We loose 5 times 128
    // bytes this way.
    If (I mod 2) = 0 Then FreeMem(P2,128);
  end;
  GetMem(P2,128);
  // This will provoke an error and a memory dump
  Freemem (P2,64);
end.
```

This is the memory dump shown when running this program:

```
Marked memory at 0040FA50 invalid
Wrong size : 128 allocated 64 freed
  0x00408708
  0x0040CB49
  0x0040C481
Call trace for block 0x0040FA50 size 128
  0x0040CB3D
  0x0040C481
```

If you use the `lineinfo` unit (or use the `-gl` switch) as well, then `heaptrc` will also give you the filenames and line-numbers of the procedures in the backtrace:

```
Marked memory at 00410DA0 invalid
Wrong size : 128 allocated 64 freed
  0x004094B8
  0x0040D8F9  main,  line 25 of heapex.pp
  0x0040D231
Call trace for block 0x00410DA0 size 128
  0x0040D8ED  main,  line 23 of heapex.pp
  0x0040D231
```

If lines without filename/line-number occur, this means there is a unit which has no debug info included.

9.3 Constants, Types and variables

The `FillExtraInfoType` is a procedural type used in the `SetExtraInfo` (139) call.

```
type
  FillExtraInfoType = procedure(p : pointer);
```

The following typed constants allow to fine-tune the standard dump of the memory usage by `DumpHeap` (138):

```
const
  tracesize = 8;
```

```
quicktrace : boolean = true;  
HaltOnError : boolean = true;  
keepreleased : boolean = false;  
add_tail : boolean = true;  
usecrc : boolean = true
```

TraceSize specifies how many levels of calls are displayed of the call stack during the memory dump. If you specify keepreleased:=True then half the TraceSize is reserved for the GetMem call stack, and the other half is reserved for the FreeMem call stack. For example, the default value of 8 will cause eight levels of call frames to be dumped for the getmem call if keepreleased is False. If KeepReleased is true, then 4 levels of call frames will be dumped for the GetMem call and 4 frames will be dumped for the FreeMem call. If you want to change this value, you must recode the heaptrc unit.

Quicktrace determines whether the memory manager checks whether a block that is about to be released is allocated correctly. This is a rather time consuming search, and slows program execution significantly, so by default it is set to False.

If HaltOnError is set to True then an illegal call to FreeMem will cause the memory manager to execute a halt(1) instruction, causing a memory dump. By Default it is set to True.

If keepreleased is set to true, then a list of freed memory blocks is kept. This is useful if you suspect that the same memory block is released twice. However, this option is very memory intensive, so use it sparingly, and only when it's really necessary.

If add_tail is True (the default) then a check is also performed on the memory location just behind the allocated memory.

If usecrc is True (the default) then a crc check is performed on locations before and after the allocated memory. This is useful to detect memory overwrites.

9.4 Functions and procedures

DumpHeap

Declaration: `procedure DumpHeap;`

Description: DumpHeap dumps to standard output a summary of memory usage. It is called automatically by the heaptrc unit when your program exits (by instaling an exit procedure), but it can be called at any time

Errors: None.

See also: MarkHeap ([138](#))

MarkHeap

Declaration: `procedure MarkHeap;`

Description: MarkHeap marks all memory blocks with a special signature. You can use this if you think that you corrupted the memory.

Errors: None.

See also: DumpHeap ([138](#))

SetExtraInfo

Declaration: `procedure SetExtraInfo(size : longint; func : FillExtraInfoType);`

Description: You can use `SetExtraInfo` to store extra info in the blocks that the `heaptrc` unit reserves when tracing `getmem` calls. `Size` indicates the size (in bytes) that the trace mechanism should reserve for your extra information. For each call to `getmem`, `func` will be called, and passed a pointer to the memory reserved.

When dumping the memory summary, the extra info is shown as `Longint` values.

Errors: You can only call `SetExtraInfo` if no memroy has been allocated yet. If memory was already allocated prior to the call to `SetExtraInfo`, then an error will be displayed on standard error output, and a `DumpHeap` ([138](#)) is executed.

See also: `DumpHeap` ([138](#)), `SetHeapTraceOutput` ([140](#))

Program `heapex;`

{ Program used to demonstrate the usage of heaptrc unit }

Uses `heaptrc;`

Var `P1 : ^Longint;`
 `P2 : Pointer;`
 `I : longint;`
 `Marker : Longint;`

Procedure `SetMarker (P : pointer);`

Type `PLongint = ^Longint;`

begin
 `PLongint(P)^:= Marker;`
end;

Procedure `Part1;`

begin
 // Blocks allocated here are marked with \$FFAAFFAA = -5570646
 `Marker := $FFAAFFAA;`
 `New(P1);`
 `New(P1);`
 `Dispose(P1);`
 For `I:=1 to 10 do`
 begin
 `GetMem (P2,128);`
 If `(I mod 2) = 0` **Then** `FreeMem(P2,128);`
 end;
 `GetMem(P2,128);`
 end;

Procedure `Part2;`

begin
 // Blocks allocated here are marked with \$FAFAFAFA = -84215046
 `Marker := $FAFAFAFA;`
 `New(P1);`
 `New(P1);`
 `Dispose(P1);`

```
For I:=1 to 10 do
  begin
    GetMem (P2,128);
    If (I mod 2) = 0 Then FreeMem(P2,128);
  end;
  GetMem(P2,128);
end;

begin
  SetExtraInfo (SizeOf (Marker), @SetMarker);
  Writeln ( 'Part 1' );
  part1;
  Writeln ( 'Part 2' );
  part2;
end.
```

SetHeapTraceOutput

Declaration: `Procedure SetHeapTraceOutput(const name : string);`

Description: `SetHeapTraceOutput` sets the filename into which heap trace info will be written. By default information is written to standard output, this function allows you to redirect the information to a file with full filename name.

Errors: If the file cannot be written to, errors will occur when writing the trace.

See also: `SetExtraInfo` ([139](#))

Chapter 10

The IPC unit.

This chapter describes the IPC unit for Free Pascal. It was written for LINUX by Michaël Van Canneyt. It gives all the functionality of system V Inter-Process Communication: shared memory, semaphores and messages. It works only on the LINUX operating system.

The chapter is divided in 2 sections:

- The first section lists types, constants and variables from the interface part of the unit.
- The second section describes the functions defined in the unit.

10.1 Types, Constants and variables :

Variables

```
Var
    IPCError : longint;
```

The `IPCError` variable is used to report errors, by all calls.

Constants

Many constants here are provided for completeness only, and should under normal circumstances not be used by the programmer.

```
Const
    IPC_CREAT  = 1 shl 9; { create if key is nonexistent }
    IPC_EXCL   = 2 shl 9; { fail if key exists }
    IPC_NOWAIT = 4 shl 9; { return error on wait }
```

These constants are used in the various `xxxget` calls.

```
IPC_RMID = 0;      { remove resource }
IPC_SET  = 1;      { set ipc_perm options }
IPC_STAT = 2;      { get ipc_perm options }
IPC_INFO = 3;      { see ipcs }
```

These constants can be passed to the various `xxxctl` calls.

```
const
MSG_NOERROR = 1 shl 12;
MSG_EXCEPT = 2 shl 12;
MSGMNI = 128;
MSGMAX = 4056;
MSGMNB = 16384;
```

These constants are used in the messaging system, they are not for use by the programmer.

```
const
SEM_UNDO = $1000;
GETPID = 11;
GETVAL = 12;
GETALL = 13;
GETNCNT = 14;
GETZCNT = 15;
SETVAL = 16;
SETALL = 17;
```

These constants call be specified in the `semop` ([150](#)) call.

```
SEMMNI = 128;
SEMMSL = 32;
SEMMNS = (SEMMNI * SEMMSL);
SEMOPM = 32;
SEMVMX = 32767;
```

These constanst are used internally by the semaphore system, they should not be used by the programmer.

```
const
SHM_R      = 4 shl 6;
SHM_W      = 2 shl 6;
SHM_RDONLY = 1 shl 12;
SHM_RND     = 2 shl 12;
SHM_REMAP  = 4 shl 12;
SHM_LOCK   = 11;
SHM_UNLOCK = 12;
```

These constants are used in the `shmctl` ([156](#)) call.

Types

The following two types are provided because they are needed. One they they should be defined in the system unit, however.

```
Type
PULong = ^Cardinal;
PWord  = ^Word;
```

```
Type
TKey   = Longint;
```

TKey is the type returned by the `ftok` ([146](#)) key generating function.


```
type
  PIPC_Perm = ^TIPC_Perm;
  TIPC_Perm = record
    key : TKey;
    uid,
    gid,
    cuid,
    cgid,
    mode,
    seq : Word;
  end;
```

The TIPC_Perm structure is used in all IPC systems to specify the permissions.

```
Type
  PSHMid_DS = ^TSHMid_ds;
  TSHMid_ds = record
    shm_perm : TIPC_Perm;
    shm_segsz : longint;
    shm_atime : longint;
    shm_dtime : longint;
    shm_ctime : longint;
    shm_cpid : word;
    shm_lpid : word;
    shm_nattch : integer;
    shm_npages : word;
    shm_pages : Pointer;
    attaches : pointer;
  end;
```

The TSHMid_ds structure is used in the `shmctl` ([156](#)) call to set or retrieve settings concerning shared memory.

```
type
  PSHMinfo = ^TSHMinfo;
  TSHMinfo = record
    shmmax : longint;
    shmmmin : longint;
    shmmni : longint;
    shmseg : longint;
    shmall : longint;
  end;
```

The TSHMinfo record is used by the shared memory system, and should not be accessed by the programmer directly.

```
type
  PMSG = ^TMSG;
  TMSG = record
    msg_next : PMSG;
    msg_type : Longint;
    msg_spot : PChar;
    msg_stime : Longint;
    msg_ts : Integer;
  end;
```

The TMSG record is used in the handling of message queues. There should be few cases where the programmer needs to access this data.

```
type
  PMSQid_ds = ^TMSQid_ds;
  TMSQid_ds = record
    msg_perm    : TIPC_perm;
    msg_first   : PMsg;
    msg_last    : PMsg;
    msg_stime   : Longint;
    msg_rtime   : Longint;
    msg_ctime   : Longint;
    wwait       : Pointer;
    rwait       : pointer;
    msg_cbytes  : word;
    msg_qnum    : word;
    msg_qbytes  : word;
    msg_lspid   : word;
    msg_lrpid   : word;
  end;
```

The TMSQid_ds record is returned by the `msgctl` (147) call, and contains all data about a message queue.

```
PMSGbuf = ^TMSGbuf;
TMSGbuf = record
  mtype : longint;
  mtext : array[0..0] of char;
end;
```

The TMSGbuf record is a record containing the data of a record. you should never use this record directly, instead you should make your own record that follows the structure of the TMSGbuf record, but that has a size that is big enough to accomodate your messages. The `mtype` field should always be present, and should always be filled.

```
Type
  PMSGinfo = ^TMSGinfo;
  TMSGinfo = record
    msgpool : Longint;
    msgmap  : Longint;
    msgmax  : Longint;
    msgmnb  : Longint;
    msgmni  : Longint;
    msgssz  : Longint;
    msgtql  : Longint;
    msgseg  : Word;
  end;
```

The TMSGinfo record is used internally by the message queue system, and should not be used by the programmer directly.

```
Type
  PSEMid_ds = ^PSEMid_ds;
  TSEMid_ds = record
```

```
sem_perm : tipc_perm;
sem_otime : longint;
sem_ctime : longint;
sem_base      : pointer;
sem_pending   : pointer;
sem_pending_last : pointer;
undo          : pointer;
sem_nsems    : word;
end;
```

The TSEMid_ds structure is returned by the `semctl` (151) call, and contains all data concerning a semaphore.

Type

```
PSEMbuf = ^TSEMbuf;
TSEMbuf = record
    sem_num : word;
    sem_op  : integer;
    sem_flg : integer;
end;
```

The TSEMbuf record is used in the `semop` (150) call, and is used to specify which operations you want to do.

Type

```
PSEMinfo = ^TSEMinfo;
TSEMinfo = record
    semmap : longint;
    semmni : longint;
    semmns : longint;
    semmnu : longint;
    semmsl : longint;
    semopm : longint;
    semume : longint;
    semusz : longint;
    semvmx : longint;
    semaem : longint;
end;
```

The TSEMinfo record is used internally by the semaphore system, and should not be used directly.

Type

```
PSEMun = ^TSEMun;
TSEMun = record
    case longint of
        0 : ( val : longint );
        1 : ( buf : PSEMid_ds );
        2 : ( arr : PWord );
        3 : ( padbuf : PSEminfo );
        4 : ( padpad : pointer );
    end;
```

The TSEMun variant record (actually a C union) is used in the `semctl` (151) call.

10.2 Functions and procedures

ftok

Declaration: `Function ftok (Path : String; ID : char) : TKey;`

Description: `ftok` returns a key that can be used in a `semget` (150), `shmget` (155) or `msgget` (146) call to access a new or existing IPC resource.

`Path` is the name of a file in the file system, `ID` is a character of your choice. The `ftok` call does the same as it's C counterpart, so a pascal program and a C program will access the same resource if they use the same `Path` and `ID`

Errors: `ftok` returns -1 if the file in `Path` doesn't exist.

See also: `semget` (150), `shmget` (155), `msgget` (146)

For an example, see `msgctl` (147), `semctl` (151), `shmctl` (156).

msgget

Declaration: `Function msgget(key: TKey; msgflg:longint):longint;`

Description: `msgget` returns the ID of the message queue described by `key`. Depending on the flags in `msgflg`, a new queue is created.

`msgflg` can have one or more of the following values (combined by ORs):

IPC_CREATThe queue is created if it doesn't already exist.

IPC_EXCLIf used in combination with `IPC_CREAT`, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

Errors: On error, -1 is returned, and `IPCError` is set.

See also: `ftok` (146), `msgsnd` (146), `msgrcv` (147), `msgctl` (147), `semget` (2)

For an example, see `msgctl` (147).

msgsnd

Declaration: `Function msgsnd(msqid:longint; msgp: PMSGBuf; msgsz: longint; msgflg:longint): Boolean;`

Description: `msgsnd` sends a message to a message queue with ID `msqid`. `msgp` is a pointer to a message buffer, that should be based on the `TMSGBuf` type. `msgsz` is the size of the message (NOT of the message buffer record !)

The `msgflg` can have a combination of the following values (ORed together):

0No special meaning. The message will be written to the queue. If the queue is full, then the process is blocked.

IPC_NOWAITIf the queue is full, then no message is written, and the call returns immediatly.

The function returns `True` if the message was sent successfully, `False` otherwise.

Errors: In case of error, the call returns `False`, and `IPCError` is set.

See also: `msgget` (146), `msgrcv` (147), `seefmsgctl`

For an example, see `msgctl` (147).

msgrcv

Declaration: `Function msgrcv(msqid:longint; msgp: PMSGBuf; msgsz: longint; msgtyp:longint; msgflg:longint): Boolean;`

Description: `msgrcv` retrieves a message of type `msgtyp` from the message queue with ID `msqid`. `msgtyp` corresponds to the `mtype` field of the `TMSGbuf` record. The message is stored in the `MSGbuf` structure pointed to by `msgp`.

The `msgflg` parameter can be used to control the behaviour of the `msgrcv` call. It consists of an ORed combination of the following flags:

0No special meaning.

IPC_NOWAITIf no messages are available, then the call returns immediatly, with the `ENOMSG` error.

MSG_NOERRORIf the message size is wrong (too large), no error is generated, instead the message is truncated. Normally, in such cases, the call returns an error (`E2BIG`)

The function returns `True` if the message was received correctly, `False` otherwise.

Errors: In case of error, `False` is returned, and `IPCError` is set.

See also: `msgget` (146), `msgsnd` (146), `msgctl` (147)

For an example, see `msgctl` (147).

msgctl

Declaration: `Function msgctl(msqid:longint; cmd: longint; buf: PMSQid_ds): Boolean;`

Description: `msgctl` performs various operations on the message queue with id `ID`. Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STATIn this case, the `msgctl` call fills the `TMSQid_ds` structure with information about the message queue.

IPC_SETIn this case, the `msgctl` call sets the permissions of the queue as specified in the `ipc_perm` record inside `buf`.

IPC_RMIDIf this is specified, the message queue will be removed from the system.

`buf` contains the data that are needed by the call. It can be `Nil` in case the message queue should be removed.

The function returns `True` if successfull, `False` otherwise.

Errors: On error, `False` is returned, and `IPCError` is set accordingly.

See also: `msgget` (146), `msgsnd` (146), `msgrcv` (147)

```
program msgtool;

Uses ipc;

Type
  PMyMsgBuf = ^TMyMsgBuf;
  TMyMsgBuf = record
    mtype : Longint;
    mtext : string[255];
  end;

Procedure DoError (Const Msg : string);

begin
  WriteLn (msg,'returned an error : ',ipccerror);
  halt(1);
end;

Procedure SendMessage (Id : Longint;
  Var Buf : TMyMsgBuf;
  MType : Longint;
  Const MText : String);

begin
  WriteLn ( 'Sending message.' );
  Buf.mtype:=mtype;
  Buf.Mtext:=mtext;
  If not msgsnd(Id,PMsgBuf(@Buf),256,0) then
    DoError('msgsnd');
end;

Procedure ReadMessage (ID : Longint;
  Var Buf : TMyMsgBuf;
  MType : longint);

begin
  WriteLn ( 'Reading message.' );
  Buf.MType:=MType;
  If msgrcv(ID,PMSGBuf(@Buf),256,mtype,0) then
    WriteLn ( 'Type : ',buf.mtype,' Text : ',buf.mtext)
  else
    DoError ( 'msgrcv' );
end;

Procedure RemoveQueue ( ID : Longint);

begin
  If msgctl (id,IPC_RMID,Nil) then
    WriteLn ( 'Removed Queue with id',Id);
end;

Procedure ChangeQueueMode (ID,mode : longint);

Var QueueDS : TMSQid_ds;

begin
  If Not msgctl (Id,IPC_STAT,@QueueDS) then
    DoError ( 'msgctl : stat' );
```

```
Writeln ( 'Old permissions : ', QueueDS.msg_perm.mode);
QueueDS.msg_perm.mode:=Mode;
if msgctl ( ID,IPC_SET,@QueueDS) then
  Writeln ( 'New permissions : ', QueueDS.msg_perm.mode)
else
  DoError ( 'msgctl : IPC_SET' );
end;

procedure usage;

begin
  Writeln ( 'Usage : msgtool s(end) <type> <text> (max 255 characters)');
  Writeln ( '                                r(eceive) <type>');
  Writeln ( '                                d(DELETE)');
  Writeln ( '                                m(ode) <decimal mode>');
  halt(1);
end;

Function StrToInt (S : String): longint;

Var M : longint;
    C : Integer;

begin
  val (S,M,C);
  If C<>0 Then DoError ( ' StrToInt : '+S);
  StrToInt:=M;
end;

Var
  Key : TKey;
  ID : longint;
  Buf : TMyMsgBuf;

begin
  If Paramcount<1 then Usage;
  key := Ftok ( '.', 'M' );
  ID:=msgget(key,IPC_CREAT or 438);
  If ID<0 then DoError ( 'MsgGet' );
  Case upCase(Paramstr(1)[1]) of
    'S' : If ParamCount<>3 then
      Usage
    else
      SendMessage ( id, Buf, StrToInt (Paramstr(2)), paramstr(3));
    'R' : If ParamCount<>2 then
      Usage
    else
      ReadMessage ( id, buf, strtoint (Paramstr(2)));
    'D' : If ParamCount<>1 then
      Usage
    else
      RemoveQueue ( ID );
    'M' : If ParamCount<>2 then
      Usage
    else
      ChangeQueueMode ( id, strtoint (paramstr(2)));
  else
    Usage
  end;
```

```
end;  
end.
```

semget

Declaration: `Function semget(key:Tkey; nsems:longint; semflg:longint): longint;`

Description: `semget` returns the ID of the semaphore set described by `key`. Depending on the flags in `semflg`, a new queue is created.

`semflg` can have one or more of the following values (combined by ORs):

IPC_CREATThe queue is created if it doesn't already exist.

IPC_EXCLIf used in combination with `IPC_CREAT`, causes the call to fail if the set already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

if a new set of semaphores is created, then there will be `nsems` semaphores in it.

Errors: On error, -1 is returned, and `IPCError` is set.

See also: `ftok` ([146](#)), `semop` ([150](#)), `semctl` ([151](#))

semop

Declaration: `Function semop(semid:longint; sops: pointer; nsops: cardinal): Boolean;`

Description: `semop` performs a set of operations on a message queue. `sops` points to an array of type `TSEMbuf`. The array should contain `nsops` elements.

The fields of the `TSEMbuf` structure

```
TSEMbuf = record  
    sem_num : word;  
    sem_op  : integer;  
    sem_flg : integer;
```

should be filled as follows:

sem_numThe number of the semaphore in the set on which the operation must be performed.

sem_opThe operation to be performed. The operation depends on the sign of `sem_op`

- 1.A positive number is simply added to the current value of the semaphore.
- 2.If 0 (zero) is specified, then the process is suspended until the specified semaphore reaches zero.
- 3.If a negative number is specified, it is subtracted from the current value of the semaphore. If the value would become negative then the process is suspended until the value becomes big enough, unless `IPC_NOWAIT` is specified in the `sem_flg`.

sem_flgOptional flags: if `IPC_NOWAIT` is specified, then the calling process will never be suspended.

The function returns `True` if the operations were successful, `False` otherwise.

Errors: In case of error, `False` is returned, and `IPCError` is set.

See also: `semget` ([150](#)), `semctl` ([151](#))

semctl

Declaration: `Function semctl(semid:longint; semnum:longint; cmd:longint; var arg:tsemun): longint;`

Description: `semctl` performs various operations on the semaphore `semnum` with semaphore set id `ID`.

The `arg` parameter supplies the data needed for each call. This is a variant record that should be filled differently, according to the command:

```
Type
  TSEMun = record
    case longint of
      0 : ( val : longint );
      1 : ( buf : PSEMid_ds );
      2 : ( arr : PWord );
      3 : ( padbuf : PSeminfo );
      4 : ( padpad : pointer );
    end;
```

Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STATIn this case, the `arg` record should have its `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call fills this `TSEMid_ds` structure with information about the semaphore set.

IPC_SETIn this case, the `arg` record should have its `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call sets the permissions of the queue as specified in the `ipc_perm` record.

IPC_RMIDIf this is specified, the semaphore set is removed from the system.

GETALLIn this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be stored. The size of this memory area is `SizeOf(Word) * Number of semaphores in the set`. This call will then fill the memory array with all the values of the semaphores.

GETNCNTThis will fill the `val` field of the `arg` union with the number of processes waiting for resources.

GETPID`semctl` returns the process ID of the process that performed the last `semop` (150) call.

GETVAL`semctl` returns the value of the semaphore with number `semnum`.

GETZCNT`semctl` returns the number of processes waiting for semaphores that reach value zero.

SETALLIn this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be retrieved from. The size of this memory area is `SizeOf(Word) * Number of semaphores in the set`. This call will then set the values of the semaphores from the memory array.

SETVALThis will set the value of semaphore `semnum` to the value in the `val` field of the `arg` parameter.

The function returns -1 on error.

Errors: The function returns -1 on error, and `IPCError` is set accordingly.

See also: `semget` (150), `semop` (150)

```
Program semtool;

{ Program to demonstrat the use of semaphores }

Uses ipc;

Const MaxSemValue = 5;

Procedure DoError (Const Msg : String);

begin
  WriteLn ( 'Error : ',msg, ' Code : ',IPCError);
  Halt(1);
end;

Function getsemval (ID,Member : longint) : longint;

Var S : TSEMun;

begin
  GetSemVal:=SemCtl(id ,member,GETVAL,S);
end;

Procedure DispVal (ID,member : longint);

begin
  writeln ( 'Value for member ',member, ' is ',GetSemVal(ID,Member));
end;

Function GetMemberCount (ID : Longint) : longint;

Var opts : TSEMun;
    semds : TSEMids;

begin
  opts.buf:=@semds;
  If semctl(Id,0,IPC_STAT,opts)<>-1 then
    GetMemberCount:=semds.sem_nsems
  else
    GetMemberCount:=-1;
end;

Function OpenSem (Key : TKey) : Longint;

begin
  OpenSem:=semget(Key,0,438);
  If OpenSem=-1 then
    DoError ( 'OpenSem' );
end;

Function CreateSem (Key : TKey; Members : Longint) : Longint;

Var Count : Longint;
    Semopts : TSemun;

begin
  If members>semmsl then
    DoError ( 'Sorry, maximum number of semaphores in set exceeded');
```

```
Writeln ('Trying to create a new semaphore set with ',members,' members.');
```

```
CreateSem:=semget(key,members,IPC_CREAT or IPC_Excl or 438);
```

```
If CreateSem=-1 then
```

```
    DoError ('Semaphore set already exists.');
```

```
Semopts.val:=MaxSemValue; { Initial value of semaphores }
```

```
For Count:=0 to Members-1 do
```

```
    semctl(CreateSem,count,setval,semopts);
```

```
end;
```

```
Procedure lockSem (ID,Member: Longint);
```

```
Var lock : TSEMbuf;
```

```
begin
```

```
    With lock do
```

```
        begin
```

```
            sem_num:=0;
```

```
            sem_op:=-1;
```

```
            sem_flg:=IPC_NOWAIT;
```

```
        end;
```

```
    if (member<0) or (member>GetMemberCount(ID)-1) then
```

```
        DoError ('semaphore member out of range');
```

```
    if getsemval(ID,member)=0 then
```

```
        DoError ('Semaphore resources exhausted (no lock)');
```

```
    lock.sem_num:=member;
```

```
    Writeln ('Attempting to lock member ',member, ' of semaphore ',ID);
```

```
    if not semop(Id,@lock,1) then
```

```
        DoError ('Lock failed')
```

```
    else
```

```
        Writeln ('Semaphore resources decremented by one');
```

```
        dispval(ID,Member);
```

```
end;
```

```
Procedure UnlockSem (ID,Member: Longint);
```

```
Var Unlock : TSEMbuf;
```

```
begin
```

```
    With Unlock do
```

```
        begin
```

```
            sem_num:=0;
```

```
            sem_op:=1;
```

```
            sem_flg:=IPC_NOWAIT;
```

```
        end;
```

```
    if (member<0) or (member>GetMemberCount(ID)-1) then
```

```
        DoError ('semaphore member out of range');
```

```
    if getsemval(ID,member)=MaxSemValue then
```

```
        DoError ('Semaphore not locked');
```

```
    Unlock.sem_num:=member;
```

```
    Writeln ('Attempting to unlock member ',member, ' of semaphore ',ID);
```

```
    if not semop(Id,@unlock,1) then
```

```
        DoError ('Unlock failed')
```

```
    else
```

```
        Writeln ('Semaphore resources incremented by one');
```

```
        dispval(ID,Member);
```

```
end;
```

```
Procedure RemoveSem (ID : longint);
```

```
var S : TSemun;

begin
  If semctl(Id,0,IPC_RMID,s)<>-1 then
    Writeln ( 'Semaphore removed')
  else
    DoError ( 'Couldn''t remove semaphore');
end;

Procedure ChangeMode (ID,Mode : longint);

Var rc : longint;
    opts : TSEMun;
    semds : TSEMid_ds;

begin
  opts.buf:=@semds;
  If not semctl (Id,0,IPC_STAT,opts)<>-1 then
    DoError ( 'Couldn''t stat semaphore');
  Writeln ( 'Old permissions were : ',semds.sem_perm.mode);
  semds.sem_perm.mode:=mode;
  If semctl(id,0,IPC_SET,opts)<>-1 then
    Writeln ( 'Set permissions to ',mode)
  else
    DoError ( 'Couldn''t set permissions');
end;

Procedure PrintSem (ID : longint);

Var l,cnt : longint;

begin
  cnt:=getmembercount(ID);
  Writeln ( 'Semaphore ',ID,' has ',cnt,' Members');
  For l:=0 to cnt-1 Do
    DispVal(id,i);
end;

Procedure USage;

begin
  Writeln ( 'Usage : semtool c(reate) <count>');
  Writeln ( '                l(ock) <member>');
  Writeln ( '                u(nlock) <member>');
  Writeln ( '                d(elte)');
  Writeln ( '                m(ode) <mode>');
  halt(1);
end;

Function StrToInt (S : String): longint;

Var M : longint;
    C : Integer;

begin
  val (S,M,C);
```

```
    If C<>0 Then DoError ( ' StrToInt : '+S);
    StrToInt:=M;
end;

Var Key : Tkey;
    ID : Longint;

begin
    If ParamCount<1 then Usage;
    key:=ftok ( '.' , 's' );
    Case UpCase(Paramstr(1))[1] of
        'C' : begin
                if paramcount<>2 then usage;
                CreateSem ( key, strtoint(paramstr(2)));
            end;
        'L' : begin
                if paramcount<>2 then usage;
                ID:=OpenSem ( key);
                LockSem ( ID, strtoint(paramstr(2)));
            end;
        'U' : begin
                if paramcount<>2 then usage;
                ID:=OpenSem ( key);
                UnLockSem ( ID, strtoint(paramstr(2)));
            end;
        'M' : begin
                if paramcount<>2 then usage;
                ID:=OpenSem ( key);
                ChangeMode ( ID, strtoint(paramstr(2)));
            end;
        'D' : Begin
                ID:=OpenSem(Key);
                RemoveSem(Id);
            end;
        'P' : begin
                ID:=OpenSem(Key);
                PrintSem( Id );
            end;
    else
        Usage
    end;
end.
```

shmget

Declaration: Function shmget(key: Tkey; Size:longint; flag:longint):longint;

Description: shmget returns the ID of a shared memory block, described by key. Depending on the flags in flag, a new memory block is created.

flag can have one or more of the following values (combined by ORs):

IPC_CREATThe queue is created if it doesn't already exist.

IPC_EXCLIf used in combination with IPC_CREAT, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

if a new memory block is created, then it will have size `Size` semaphores in it.

Errors: On error, -1 is returned, and `IPCError` is set.

See also:

shmat

Declaration: `Function shmat (shmid:longint; shmaddr:pchar; shmflg:longint):pchar;`

Description: `shmat` attaches a shared memory block with identified `shmid` to the current process. The function returns a pointer to the shared memory block.

If `shmaddr` is `Nil`, then the system chooses a free unmapped memory region, as high up in memory space as possible.

If `shmaddr` is non-nil, and `SHM_RND` is in `shmflg`, then the returned address is `shmaddr`, rounded down to `SHMLBA`. If `SHM_RND` is not specified, then `shmaddr` must be a page-aligned address.

The parameter `shmflg` can be used to control the behaviour of the `shmat` call. It consists of a ORed combination of the following constants:

SHM_RNDThe suggested address in `shmaddr` is rounded down to `SHMLBA`.

SHM_RDONLYthe shared memory is attached for read access only. Otherwise the memory is attached for read-write. The process then needs read-write permissions to access the shared memory.

Errors: If an error occurs, -1 is returned, and `IPCError` is set.

See also: `shmget` ([155](#)), `shmdt` ([156](#)), `shmctl` ([156](#))

For an example, see `shmctl` ([156](#)).

shmdt

Declaration: `Function shmdt (shmaddr:pchar):boolean;`

Description: `shmdt` detaches the shared memory at address `shmaddr`. This shared memory block is unavailable to the current process, until it is attached again by a call to `shmat` ([156](#)).

The function returns `True` if the memory block was detached successfully, `False` otherwise.

Errors: On error, `False` is returned, and `IPCError` is set.

See also: `shmget` ([155](#)), `shmat` ([156](#)), `shmctl` ([156](#))

shmctl

Declaration: `Function shmctl(shmid:longint; cmd:longint; buf: TSHMid_ds): Boolean;`

Description: `shmctl` performs various operations on the shared memory block identified by identifier `shmid`.

The `buf` parameter points to a `TSHMid_ds` record. The `cmd` parameter is used to pass which operation is to be performed. It can have one of the following values :

IPC_STAT`shmctl` fills the `TSHMid_ds` record that `buf` points to with the available information about the shared memory block.

IPC_SET applies the values in the `ipc_perm` record that `buf` points to, to the shared memory block.

IPC_RMID the shared memory block is destroyed (after all processes to which the block is attached, have detached from it).

If successful, the function returns `True`, `False` otherwise.

Errors: If an error occurs, the function returns `False`, and `IPCerror` is set.

See also: `shmget` ([155](#)), `shmat` ([156](#)), `shmdt` ([156](#))

```
Program shmtool;

uses ipc, strings;

Const SegSize = 100;

var key : Tkey;
    shmId, cntr : longint;
    segptr : pchar;

Procedure USage;

begin
  Writeln ( 'Usage : shmtool w(rite) text' );
  writeln ( '                r(ead)' );
  writeln ( '                d(elete)' );
  writeln ( '                m(ode change) mode' );
  halt(1);
end;

Procedure Writeshm (ID : Longint; ptr : pchar; S : string);

begin
  strcpy ( ptr, S );
end;

Procedure Readshm (ID : longint; ptr : pchar);

begin
  Writeln ( 'Read : ', ptr );
end;

Procedure removeshm (ID : Longint);

begin
  shmctl (ID, IPC_RMID, Nil);
  writeln ( 'Shared memory marked for deletion' );
end;

Procedure CHangeMode (ID : longint; mode : String);

Var m : word;
    code : integer;
    data : TSHMid_ds;

begin
  val (mode, m, code);
  if code <> 0 then
```

```
usage;
If Not shmctl (shmctl,IPC_STAT,@data) then
  begin
    writeln ('Error : shmctl :',ipcerror);
    halt(1);
  end;
  writeln ('Old permissions : ',data.shm_perm.mode);
  data.shm_perm.mode:=m;
  If Not shmctl (shmctl,IPC_SET,@data) then
    begin
      writeln ('Error : shmctl :',ipcerror);
      halt(1);
    end;
    writeln ('New permissions : ',data.shm_perm.mode);
end;

begin
  if paramcount<1 then usage;
  key := ftok ('.', 'S');
  shmctl := shmget(key,segsz,IPC_CREAT or IPC_EXCL or 438);
  If shmctl=-1 then
    begin
      writeln ('Shared memory exists. Opening as client');
      shmctl := shmget(key,segsz,0);
      If shmctl = -1 then
        begin
          writeln ('shmget : Error !',ipcerror);
          halt(1);
        end
      end
    else
      writeln ('Creating new shared memory segment. ');
      segptr:=shmat(shmctl,nil,0);
      if longint(segptr)=-1 then
        begin
          writeln ('Shmat : error !',ipcerror);
          halt(1);
        end;
      case upcase(paramstr(1)[1]) of
        'W' : writeshm (shmctl,segptr,paramstr(2));
        'R' : readshm (shmctl,segptr);
        'D' : removeshm(shmctl);
        'M' : changemode (shmctl,paramstr(2));
      else
        begin
          writeln (paramstr(1));
          usage;
        end;
      end;
    end;
end.
```

Chapter 11

The KEYBOARD unit

The `KeyBoard` unit implements a keyboard access layer which is system independent. It can be used to poll the keyboard state and wait for certain events. Waiting for a keyboard event can be done with the `GetKeyEvent` (163) function, which will return a driver-dependent key event. This key event can be translated to a interpretable event by the `TranslateKeyEvent` (170) function. The result of this function can be used in the other event examining functions.

A custom keyboard driver can be installed using the `SetKeyboardDriver` (169) function. The current keyboard driver can be retrieved using the `GetKeyboardDriver` (163) function. The last section of this chapter demonstrates how to make a keyboard driver.

11.1 Constants, Type and variables

Constants

The following constants define some error constants, which may be returned by the keyboard functions.

```
errKbdBase          = 1010;
errKbdInitError      = errKbdBase + 0;
errKbdNotImplemented = errKbdBase + 1;
```

The following constants denote special keyboard keys. The first constants denote the function keys:

```
const
  kbdF1      = $FF01;
  kbdF2      = $FF02;
  kbdF3      = $FF03;
  kbdF4      = $FF04;
  kbdF5      = $FF05;
  kbdF6      = $FF06;
  kbdF7      = $FF07;
  kbdF8      = $FF08;
  kbdF9      = $FF09;
  kbdF10     = $FF0A;
  kbdF11     = $FF0B;
  kbdF12     = $FF0C;
  kbdF13     = $FF0D;
  kbdF14     = $FF0E;
```

```
kbdF15      = $FF0F;
kbdF16      = $FF10;
kbdF17      = $FF11;
kbdF18      = $FF12;
kbdF19      = $FF13;
kbdF20      = $FF14;
```

Constants \$15 till \$1F are reserved for future function keys. The following constants denote the cursor movement keys:

```
kbdHome     = $FF20;
kbdUp       = $FF21;
kbdPgUp     = $FF22;
kbdLeft     = $FF23;
kbdMiddle   = $FF24;
kbdRight    = $FF25;
kbdEnd      = $FF26;
kbdDown     = $FF27;
kbdPgDn     = $FF28;

kbdInsert   = $FF29;
kbdDelete   = $FF2A;
```

Constants \$2B till \$2F are reserved for future keypad keys. The following flags are also defined:

```
kbASCII     = $00;
kbUnicode   = $01;
kbFnKey     = $02;
kbPhys      = $03;
kbReleased  = $04;
```

They can be used to check what kind of data a key event contains. The following shift-state flags can be used to determine the shift state of a key (i.e. which of the SHIFT, ALT and CTRL keys were pressed simultaneously with a key):

```
kbLeftShift = 1;
kbRightShift = 2;
kbShift     = kbLeftShift or kbRightShift;
kbCtrl      = 4;
kbAlt       = 8;
```

The following constant strings are used in the key name functions `FunctionKeyName` (162) and `KeyEventToString` (167):

```
SShift      : Array [1..3] of string[5] = ('SHIFT', 'CTRL', 'ALT');
LeftRight   : Array [1..2] of string[5] = ('LEFT', 'RIGHT');
UnicodeChar : String = 'Unicode character ';
SScanCode   : String = 'Key with scancode ';
SUnknownFunctionKey : String = 'Unknown function key : ';
SAnd        : String = 'AND';
SKeyPad     : Array [0..($FF2F-kbdHome)] of string[6] =
  ('Home', 'Up', 'PgUp', 'Left',
   'Middle', 'Right', 'End', 'Down',
   'PgDn', 'Insert', 'Delete', '',
   '', '', '', '');
```

They can be changed to localize the key names when needed.

Types

The `TKeyEvent` type is the base type for all keyboard events:

```
TKeyEvent = Longint;
```

The key stroke is encoded in the 4 bytes of the `TKeyEvent` type. The various fields of the key stroke encoding can be obtained by typecasting the `TKeyEvent` type to the `TKeyRecord` type:

```
TKeyRecord = packed record
  KeyCode : Word;
  ShiftState, Flags : Byte;
end;
```

The structure of a `TKeyRecord` structure is explained in table (11.1). The shift-state can be

Table 11.1: Structure of `TKeyRecord`

Field	Meaning
KeyCode	Depending on <code>flags</code> either the physical representation of a key (under DOS scancode, ascii code pair), or the translated ASCII/unicode character.
ShiftState	Shift-state when this key was pressed (or shortly after)
Flags	Determine how to interpret <code>KeyCode</code>

checked using the various shift-state constants, and the flags in the last byte can be checked using one of the `kbASCII`, `kbUnicode`, `kbFnKey`, `kbPhys`, `kbReleased` constants.

If there are two keys returning the same char-code, there's no way to find out which one was pressed (Gray+ and Simple+). If it needs to be known which was pressed, the untranslated keycodes must be used, but these are system dependent. System dependent constants may be defined to cover those, with possibly having the same name (but different value).

The `TKeyboardDriver` record can be used to install a custom keyboard driver with the `SetKeyboardDriver` (169) function:

Type

```
TKeyboardDriver = Record
  InitDriver : Procedure;
  DoneDriver : Procedure;
  GetKeyEvent : Function : TKeyEvent;
  PollKeyEvent : Function : TKeyEvent;
  GetShiftState : Function : Byte;
  TranslateKeyEvent : Function (KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUnicode: Function (KeyEvent: TKeyEvent): TKeyEvent;
end;
```

The various fields correspond to the different functions of the keyboard unit interface. For more information about this record see section 11.4, page 171

11.2 Functions and Procedures

DoneKeyboard

Declaration: `Procedure DoneKeyboard;`

Description: `DoneKeyboard` de-initializes the keyboard interface if the keyboard driver is active. If the keyboard driver is not active, the function does nothing.

This will cause the keyboard driver to clear up any allocated memory, or restores the console or terminal the program was running in to its initial state before the call to `InitKeyBoard` (166). This function should be called on program exit. Failing to do so may leave the terminal or console window in an unusable state. Its exact action depends on the platform on which the program is running.

Errors: None.

See also: `InitKeyBoard` (166)

For an example, see most other functions.

FunctionKeyName

Declaration: `Function FunctionKeyName (KeyCode : Word) : String;`

Description: `FunctionKeyName` returns a string representation of the function key with code `KeyCode`. This can be an actual function key, or one of the cursor movement keys.

Errors: In case `KeyCode` does not contain a function code, the `SUnknownFunctionKey` string is returned, appended with the `KeyCode`.

See also: `ShiftStateToString` (170) `KeyEventToString` (167)

Listing: `kbdex/ex8.pp`

Program `Example8;`

{ Program to demonstrate the FunctionKeyName function. }

Uses `keyboard;`

Var

`K : TKeyEvent;`

begin

`InitKeyboard;`

`WriteIn('Press function keys, press "q" to end.');`

Repeat

`K:=GetKeyEvent;`

`K:=TranslateKeyEvent(K);`

If `IsFunctionKey(k)` **then**

begin

`Write('Got function key : ');`

`WriteIn(FunctionKeyName(TkeyRecord(K).KeyCode));`

end;

Until `(GetKeyEventChar(K)='q');`

`DoneKeyboard;`

end.

GetKeyboardDriver

Declaration: `Procedure GetKeyboardDriver (Var Driver : TKeyboardDriver);`

Description: `GetKeyboardDriver` returns in `Driver` the currently active keyboard driver. This function can be used to enhance an existing keyboard driver.

For more information on getting and setting the keyboard driver section [11.4](#), page [171](#).

Errors: None.

See also: `SetKeyboardDriver` ([169](#))

GetKeyEvent

Declaration: `function GetKeyEvent: TKeyEvent;`

Description: `GetKeyEvent` returns the last keyevent if one was stored in `PendingKeyEvent`, or waits for one if none is available. A non-blocking version is available in `PollKeyEvent` ([167](#)).

The returned key is encoded as a `TKeyEvent` type variable, and is normally the physical key scan code, (the scan code is driver dependent) which can be translated with one of the translation functions `TranslateKeyEvent` ([170](#)) or `TranslateKeyEventUnicode` ([170](#)). See the types section for a description of how the key is described.

Errors: If no key became available, 0 is returned.

See also: `PutKeyEvent` ([168](#)), `PollKeyEvent` ([167](#)), `TranslateKeyEvent` ([170](#)), `TranslateKeyEventUnicode` ([170](#))

Listing: kbdex/ex1.pp

```
program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end. ');
  Repeat
    K:=GetKeyEvent;
    Write('Got key event with ');
    Case GetKeyEventFlags(K) of
      kbASCII      : Writeln('ASCII key');
      kbUnicode    : Writeln('Unicode key');
      kbFnKey      : Writeln('Function key');
      kbPhys       : Writeln('Physical key');
      kbReleased   : Writeln('Released key event');
    end;
    K:=TranslateKeyEvent(K);
    Writeln('Got key : ', KeyEventToString(K));
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

GetKeyEventChar

Declaration: `function GetKeyEventChar(KeyEvent: TKeyEvent): Char;`

Description: `GetKeyEventChar` returns the charcode part of the given `KeyEvent`, if it contains a translated character key keycode. The charcode is simply the ascii code of the character key that was pressed.

It returns the null character if the key was not a character key, but e.g. a function key.

Errors: None.

See also: `GetKeyEventUnicode` (166), `GetKeyEventShiftState` (165), `GetKeyEventFlags` (165), `GetKeyEventCode` (164), `GetKeyEvent` (163)

For an example, see `GetKeyEvent` (163)

GetKeyEventCode

Declaration: `function GetKeyEventCode(KeyEvent: TKeyEvent): Word;`

Description: `GetKeyEventCode` returns the translated function keycode part of the given `KeyEvent`, if it contains a translated function key.

If the key pressed was not a function key, the null character is returned.

Errors: None.

See also: `GetKeyEventUnicode` (166), `GetKeyEventShiftState` (165), `GetKeyEventFlags` (165), `GetKeyEventChar` (164), `GetKeyEvent` (163)

Listing: `kbdex/ex2.pp`

Program `Example2;`

{ Program to demonstrate the GetKeyEventCode function. }

Uses `keyboard;`

Var

`K : TKeyEvent;`

begin

`InitKeyBoard;`

`WriteLn('Press function keys, or press "q" to end.');`

Repeat

`K:=GetKeyEvent;`

`K:=TranslateKeyEvent(K);`

If `(GetKeyEventFlags(K)<>KbfnKey)` **then**

`WriteLn('Not a function key')`

else

begin

`Write('Got key ',GetKeyEventCode(K));`

`WriteLn('') : ', KeyEventToString(K));`

end;

Until `(GetKeyEventChar(K)='q');`

`DoneKeyboard;`

end.

GetKeyEventFlags

Declaration: `function GetKeyEventFlags(KeyEvent: TKeyEvent): Byte;`

Description: `GetKeyEventFlags` returns the flags part of the given `KeyEvent`.

Errors: None.

See also: `GetKeyEventUnicode` (166), `GetKeyEventShiftState` (165), `GetKeyEventCode` (164), `GetKeyEventChar` (164), `GetKeyEvent` (163)

For an example, see `GetKeyEvent` (163)

GetKeyEventShiftState

Declaration: `function GetKeyEventShiftState(KeyEvent: TKeyEvent): Byte;`

Description: `GetKeyEventShiftState` returns the shift-state values of the given `KeyEvent`. This can be used to detect which of the modifier keys `Shift`, `Alt` or `Ctrl` were pressed. If none were pressed, zero is returned.

Note that this function does not always return expected results; In a unix X-Term, the modifier keys do not always work.

Errors: None.

See also: `GetKeyEventUnicode` (166), `GetKeyEventFlags` (165), `GetKeyEventCode` (164), `GetKeyEventChar` (164), `GetKeyEvent` (163)

Listing: kbdex/ex3.pp

Program Example3;

{ Program to demonstrate the GetKeyEventShiftState function. }

Uses keyboard;

Var

 K : TKeyEvent;
 S : Byte;

begin

```
InitKeyBoard;  
Write(' Press keys combined with CTRL/SHIFT/ALT ');  
Writeln(' , or press "q" to end. ');  
Repeat  
    K:=GetKeyEvent;  
    K:=TranslateKeyEvent(K);  
    S:=GetKeyEventShiftState(K);  
    If (S=0) then  
        Writeln('No special keys pressed')  
    else  
        begin  
            Writeln('Detected special keys : ', ShiftStateToString(K, False));  
            Writeln('Got key : ', KeyEventToString(K));  
        end;  
    Until (GetKeyEventChar(K)= 'q');  
DoneKeyboard;  
end.
```

GetKeyEventUnicode

Declaration: `function GetKeyEventUnicode(KeyEvent: TKeyEvent): Word;`

Description: `GetKeyEventUnicode` returns the unicode part of the given `KeyEvent` if it contains a translated unicode character.

Errors: None.

See also: `GetKeyEventShiftState` ([165](#)), `GetKeyEventFlags` ([165](#)), `GetKeyEventCode` ([164](#)), `GetKeyEventChar` ([164](#)), `GetKeyEvent` ([163](#))

No example available yet.

InitKeyBoard

Declaration: `procedure InitKeyboard;`

Description: `InitKeyboard` initializes the keyboard driver. If the driver is already active, it does nothing. When the driver is initialized, it will do everything necessary to ensure the functioning of the keyboard, including allocating memory, initializing the terminal etc.

This function should be called once, before using any of the keyboard functions. When it is called, the `DoneKeyboard` ([162](#)) function should also be called before exiting the program or changing the keyboard driver with `SetKeyboardDriver` ([169](#)).

Errors: None.

See also: `DoneKeyboard` ([162](#)), `SetKeyboardDriver` ([169](#))

For an example, see most other functions.

IsFunctionKey

Declaration: `function IsFunctionKey(KeyEvent: TKeyEvent): Boolean;`

Description: `IsFunctionKey` returns `True` if the given key event in `KeyEvent` was a function key or not.

Errors: None.

See also: `GetKeyEvent` ([163](#))

Listing: `kbdex/ex7.pp`

```
program example1;  
  
  { This program demonstrates the GetKeyEvent function }  
  
uses keyboard;  
  
Var  
  K : TKeyEvent;  
  
begin  
  InitKeyBoard;  
  WriteLn('Press keys, press "q" to end. ');  
  Repeat  
    K:=GetKeyEvent;
```



```
K:=TranslateKeyEvent(K);
If IsFunctionKey(K) then
  Writeln('Got function key : ',KeyEventToString(K))
else
  Writeln('not a function key. ');
Until (GetKeyEventChar(K)='q');
DoneKeyBoard;
end.
```

KeyEventToString

Declaration: `Function KeyEventToString(KeyEvent : TKeyEvent) : String;`

Description: `KeyEventToString` translates the key event in `KeyEvent` to a human-readable description of the pressed key. It will use the constants described in the constants section to do so.

Errors: If an unknown key is passed, the scancode is returned, prefixed with the `SScanCode` string.

See also: `FunctionKeyName` ([162](#)), `ShiftStateToString` ([170](#))

For an example, see most other functions.

PollKeyEvent

Declaration: `function PollKeyEvent: TKeyEvent;`

Description: `PollKeyEvent` checks whether a key event is available, and returns it if one is found. If no event is pending, it returns 0.

Note that this does not remove the key from the pending keys. The key should still be retrieved from the pending key events list with the `GetKeyEvent` ([163](#)) function.

Errors: None.

See also: `PutKeyEvent` ([168](#)), `GetKeyEvent` ([163](#))

Listing: `kbdex/ex4.pp`

```
program example4;

{ This program demonstrates the PollKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end. ');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
        K:=GetKeyEvent;
        K:=TranslateKeyEvent(K);
        writeln;
      end;
  until K=0;
end;
```

```
        WriteLn('Got key : ', KeyEventToString(K));
    end
else
    write(' ');
    Until ( GetKeyEventChar(K)='q' );
    DoneKeyBoard;
end.
```

PollShiftStateEvent

Declaration: `function PollShiftStateEvent: TKeyEvent;`

Description: `PollShiftStateEvent` returns the current shiftstate in a keyevent. This will return 0 if there is no key event pending.

Errors: None.

See also: `PollKeyEvent` ([167](#)), `GetKeyEvent` ([163](#))

Listing: `kbdex/ex6.pp`

```
program example6;

{ This program demonstrates the PollShiftStateEvent function }

uses keyboard;

Var
    K : TKeyEvent;

begin
    InitKeyBoard;
    WriteLn('Press keys, press "q" to end. ');
    Repeat
        K:=PollKeyEvent;
        If k<>0 then
            begin
                K:=PollShiftStateEvent;
                WriteLn('Got shift state : ', ShiftStateToString(K, False));
                // Consume the key.
                K:=GetKeyEvent;
                K:=TranslateKeyEvent(K);
            end
        else
            write(' ');
        Until ( GetKeyEventChar(K)='q' );
        DoneKeyBoard;
    end.
```

PutKeyEvent

Declaration: `procedure PutKeyEvent(KeyEvent: TKeyEvent);`

Description: `PutKeyEvent` adds the given `KeyEvent` to the input queue. Please note that depending on the implementation this can hold only one value, i.e. when calling `PutKeyEvent` multiple times, only the last pushed key will be remembered.

Errors: None

See also: [PollKeyEvent \(167\)](#), [GetKeyEvent \(163\)](#)

Listing: kbdex/ex5.pp

```
program example5;

{ This program demonstrates the PutKeyEvent function }

uses keyboard;

Var
  K,k2 : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end. ');
  K2:=0;
  Repeat
    K:=GetKeyEvent;
    If k<>0 then
      begin
        if (k2 mod 2)=0 then
          K2:=K+1
        else
          K2:=0;
        K:=TranslateKeyEvent(K);
        Writeln('Got key : ',KeyEventToString(K));
        if (K2<>0) then
          begin
            PutKeyEvent(k2);
            K2:=TranslateKeyEvent(K2);
            Writeln('Put key : ',KeyEventToString(K2))
          end
        end
      end
    Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

SetKeyboardDriver

Declaration: `Function SetKeyboardDriver (Const Driver : TKeyboardDriver) : Boolean;`

Description: SetKeyBoardDriver sets the keyboard driver to Driver, if the current keyboard driver is not yet initialized. If the current keyboard driver is initialized, then SetKeyboardDriver does nothing. Before setting the driver, the currently active driver should be disabled with a call to DoneKeyboard ([162](#)).

The function returns True if the driver was set, False if not.

For more information on setting the keyboard driver, see section [11.4](#), page [171](#).

Errors: None.

See also: [GetKeyboardDriver \(163\)](#), [DoneKeyboard \(162\)](#).

ShiftStateToString

Declaration: `Function ShiftStateToString(KeyEvent : TKeyEvent; UseLeftRight : Boolean) : String;`

Description: `ShiftStateToString` returns a string description of the shift state of the key event `KeyEvent`. This can be an empty string.

The shift state is described using the strings in the `SShift` constant.

Errors: None.

See also: `FunctionKeyName` ([162](#)), `KeyEventToString` ([167](#))

For an example, see `PollShiftStateEvent` ([168](#)).

TranslateKeyEvent

Declaration: `function TranslateKeyEvent(KeyEvent: TKeyEvent): TKeyEvent;`

Description: `TranslateKeyEvent` performs ASCII translation of the `KeyEvent`. It translates a physical key to a function key if the key is a function key, and translates the physical key to the ordinal of the ascii character if there is an equivalent character key.

Errors: None.

See also: `TranslateKeyEventUnicode` ([170](#))

For an example, see `GetKeyEvent` ([163](#))

TranslateKeyEventUnicode

Declaration: `function TranslateKeyEventUnicode(KeyEvent: TKeyEvent): TKeyEvent;`

Description: `TranslateKeyEventUnicode` performs Unicode translation of the `KeyEvent`. It is not yet implemented for all platforms.

Errors: If the function is not yet implemented, then the `ErrorCode` of the `system` unit will be set to `errKbdNotImplemented`

See also:

No example available yet.

11.3 Keyboard scan codes

Special physical keys are encoded with the DOS scan codes for these keys in the second byte of the `TKeyEvent` type. A complete list of scan codes can be found in table ([11.2](#)). This is the list of keys that is used by the default key event translation mechanism. When writing a keyboard driver, either these constants should be returned by the various key event functions, or the `TranslateKeyEvent` hook should be implemented by the driver. A list of scan codes for special keys and combinations with the `SHIFT`, `ALT` and `CTRL` keys can be found in table ([11.3](#)); They are for quick reference only.

11.4 Writing a keyboard driver

Writing a keyboard driver means that hooks must be created for most of the keyboard unit functions. The `TKeyboardDriver` record contains a field for each of the possible hooks:

```
TKeyboardDriver = Record
  InitDriver : Procedure;
  DoneDriver : Procedure;
  GetKeyEvent : Function : TKeyEvent;
  PollKeyEvent : Function : TKeyEvent;
  GetShiftState : Function : Byte;
  TranslateKeyEvent : Function (KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUnicode: Function (KeyEvent: TKeyEvent): TKeyEvent;
end;
```

The meaning of these hooks is explained below:

InitDriver Called to initialize and enable the driver. Guaranteed to be called only once. This should initialize all needed things for the driver.

DoneDriver Called to disable and clean up the driver. Guaranteed to be called after a call to `initDriver`. This should clean up all things initialized by `InitDriver`.

GetKeyEvent Called by `GetKeyEvent` (163). Must wait for and return the next key event. It should NOT store keys.

PollKeyEvent Called by `PollKeyEvent` (167). It must return the next key event if there is one. Should not store keys.

GetShiftState Called by `PollShiftStateEvent` (168). Must return the current shift state.

TranslateKeyEvent Should translate a raw key event to a correct key event, i.e. should fill in the `shiftstate` and convert function key scancodes to function key keycodes. If the `TranslateKeyEvent` is not filled in, a default translation function will be called which converts the known scancodes from the tables in the previous section to a correct keyevent.

TranslateKeyEventUnicode Should translate a key event to a unicode key representation.

Strictly speaking, only the `GetKeyEvent` and `PollKeyEvent` hooks must be implemented for the driver to function correctly.

The following unit demonstrates how a keyboard driver can be installed. It takes the installed driver, and hooks into the `GetKeyEvent` function to register and log the key events in a file. This driver can work on top of any other driver, as long as it is inserted in the `uses` clause *after* the real driver unit, and the real driver unit should set the driver record in its initialization section.

Listing: `kbdex/logkeys.pp`

```
unit logkeys;

interface

Procedure StartKeyLogging;
Procedure StopKeyLogging;
Function IsKeyLogging : Boolean;
Procedure SetKeyLogFileName(FileName : String);
```

implementation**uses** sysutils, keyboard;**var****NewKeyBoardDriver**,
OldKeyBoardDriver : TKeyboardDriver;
Active, **Logging** : Boolean;
LogFileName : **String**;
KeyLog : Text;**Function** TimeStamp : **String**;**begin** TimeStamp := **FormatDateTime**('hh:nn:ss', **Time**());**end**;**Procedure** StartKeyLogging;**begin**

Logging := True;

WriteLn(KeyLog, 'Start logging keystrokes at: ', TimeStamp);**end**;**Procedure** StopKeyLogging;**begin** **WriteLn**(KeyLog, 'Stop logging keystrokes at: ', TimeStamp);

Logging := False;

end;**Function** IsKeyLogging : Boolean;**begin**

IsKeyLogging := Logging;

end;**Function** LogGetKeyEvent : TKeyEvent;**Var**

K : TKeyEvent;

begin

K := OldkeyboardDriver.GetKeyEvent();

If Logging **then** **begin** **Write**(KeyLog, TimeStamp, ': Key event: '); **WriteLn**(KeyLog, KeyEventToString(TranslateKeyEvent(K))); **end**;

LogGetKeyEvent := K;

end;**Procedure** LogInitKeyBoard;**begin**

OldKeyBoardDriver.InitDriver();

Assign(KeyLog, logFileName);

Rewrite(KeyLog);

```
    Active := True;
    StartKeyLogging;
end;

Procedure LogDoneKeyBoard;

begin
    StopKeyLogging;
    Close(KeyLog);
    Active := False;
    OldKeyBoardDriver.DoneDriver();
end;

Procedure SetKeyLogFileName(FileName : String);

begin
    If Not Active then
        LogFileName := FileName;
end;

Initialization
    GetKeyBoardDriver(OldKeyBoardDriver);
    NewKeyBoardDriver := OldKeyBoardDriver;
    NewKeyBoardDriver.GetKeyEvent := @LogGetKeyEvent;
    NewKeyBoardDriver.InitDriver := @LogInitKeyboard;
    NewKeyBoardDriver.DoneDriver := @LogDoneKeyboard;
    LogFileName := 'keyboard.log';
    Logging := False;
    SetKeyboardDriver(NewKeyBoardDriver);
end.
```

The following program demonstrates the use of the unit:

Listing: kbdex/ex9.pp

```
program example9;

{ This program demonstrates the logkeys unit }

uses keyboard, logkeys;

Var
    K : TKeyEvent;

begin
    InitKeyBoard;
    Writeln('Press keys, press "q" to end, "s" toggles logging. ');
    Repeat
        K := GetKeyEvent;
        K := TranslateKeyEvent(K);
        Writeln('Got key : ', KeyEventToString(K));
        if GetKeyEventChar(K) = 's' then
            if IsKeyLogging then
                StopKeyLogging
            else
                StartKeyLogging;
        Until (GetKeyEventChar(K) = 'q');
    DoneKeyBoard;
end.
```

Note that with a simple extension of this unit could be used to make a driver that is capable of recording and storing a set of keyboard strokes, and replaying them at a later time, so a 'keyboard macro' capable driver. This driver could sit on top of any other driver.

Table 11.2: Physical keys scan codes

Code	Key	Code	Key	Code	Key
00	NoKey	3D	F3	70	ALT-F9
01	ALT-Esc	3E	F4	71	ALT-F10
02	ALT-Space	3F	F5	72	CTRL-PrtSc
04	CTRL-Ins	40	F6	73	CTRL-Left
05	SHIFT-Ins	41	F7	74	CTRL-Right
06	CTRL-Del	42	F8	75	CTRL-end
07	SHIFT-Del	43	F9	76	CTRL-PgDn
08	ALT-Back	44	F10	77	CTRL-Home
09	ALT-SHIFT-Back	47	Home	78	ALT-1
0F	SHIFT-Tab	48	Up	79	ALT-2
10	ALT-Q	49	PgUp	7A	ALT-3
11	ALT-W	4B	Left	7B	ALT-4
12	ALT-E	4C	Center	7C	ALT-5
13	ALT-R	4D	Right	7D	ALT-6
14	ALT-T	4E	ALT-GrayPlus	7E	ALT-7
15	ALT-Y	4F	end	7F	ALT-8
16	ALT-U	50	Down	80	ALT-9
17	ALT-I	51	PgDn	81	ALT-0
18	ALT-O	52	Ins	82	ALT-Minus
19	ALT-P	53	Del	83	ALT-Equal
1A	ALT-LftBrack	54	SHIFT-F1	84	CTRL-PgUp
1B	ALT-RgtBrack	55	SHIFT-F2	85	F11
1E	ALT-A	56	SHIFT-F3	86	F12
1F	ALT-S	57	SHIFT-F4	87	SHIFT-F11
20	ALT-D	58	SHIFT-F5	88	SHIFT-F12
21	ALT-F	59	SHIFT-F6	89	CTRL-F11
22	ALT-G	5A	SHIFT-F7	8A	CTRL-F12
23	ALT-H	5B	SHIFT-F8	8B	ALT-F11
24	ALT-J	5C	SHIFT-F9	8C	ALT-F12
25	ALT-K	5D	SHIFT-F10	8D	CTRL-Up
26	ALT-L	5E	CTRL-F1	8E	CTRL-Minus
27	ALT-SemiCol	5F	CTRL-F2	8F	CTRL-Center
28	ALT-Quote	60	CTRL-F3	90	CTRL-GreyPlus
29	ALT-OpQuote	61	CTRL-F4	91	CTRL-Down
2B	ALT-BkSlash	62	CTRL-F5	94	CTRL-Tab
2C	ALT-Z	63	CTRL-F6	97	ALT-Home
2D	ALT-X	64	CTRL-F7	98	ALT-Up
2E	ALT-C	65	CTRL-F8	99	ALT-PgUp
2F	ALT-V	66	CTRL-F9	9B	ALT-Left
30	ALT-B	67	CTRL-F10	9D	ALT-Right
31	ALT-N	68	ALT-F1	9F	ALT-end
32	ALT-M	69	ALT-F2	A0	ALT-Down
33	ALT-Comma	6A	ALT-F3	A1	ALT-PgDn
34	ALT-Period	6B	ALT-F4	A2	ALT-Ins
35	ALT-Slash	6C	ALT-F5	A3	ALT-Del
37	ALT-GreyAst	6D	ALT-F6	A5	ALT-Tab
3B	F1	6E	ALT-F7		
3C	F2	6F	ALT-F8		

Table 11.3: Special keys scan codes

Key	Code	SHIFT-Key	CTRL-Key	Alt-Key
NoKey	00			
F1	3B	54	5E	68
F2	3C	55	5F	69
F3	3D	56	60	6A
F4	3E	57	61	6B
F5	3F	58	62	6C
F6	40	59	63	6D
F7	41	5A	64	6E
F8	42	5A	65	6F
F9	43	5B	66	70
F10	44	5C	67	71
F11	85	87	89	8B
F12	86	88	8A	8C
Home	47		77	97
Up	48		8D	98
PgUp	49		84	99
Left	4B		73	9B
Center	4C		8F	
Right	4D		74	9D
end	4F		75	9F
Down	50		91	A0
PgDn	51		76	A1
Ins	52	05	04	A2
Del	53	07	06	A3
Tab	8	0F	94	A5
GreyPlus			90	4E

Chapter 12

The LINUX unit.

This chapter describes the LINUX unit for Free Pascal. The unit was written by Michaël van Canneyt. It works only on the Linux operating system. This chapter is divided in 3 sections:

- The first section lists all constants, types and variables, as listed in the interface section of the LINUX unit.
- The second section gives an overview of all available functions, grouped by category.
- The third section describes all procedures and functions in the LINUX unit.

12.1 Type, Variable and Constant declarations

Types

PGlob and TGlob are 2 types used in the Glob ([224](#)) function:

```
PGlob = ^TGlob;
TGlob = record
  Name : PChar;
  Next : PGlob;
end;
```

The following types are used in the signal-processing procedures.

```
tfpreg = record
  significand: array[0..3] of word;
  exponent: word;
end;

pfpstate = ^tfpstate;
tfpstate = record
  cw, sw, tag, ipoff, cssel, dataoff, datasel: cardinal;
  st: array[0..7] of tfpreg;
  status: cardinal;
end;

PSigContextRec = ^SigContextRec;
SigContextRec = record
```

```
gs, __gsh: word;
fs, __fsh: word;
es, __esh: word;
ds, __dsh: word;
edi: cardinal;
esi: cardinal;
ebp: cardinal;
esp: cardinal;
ebx: cardinal;
edx: cardinal;
ecx: cardinal;
eax: cardinal;
trapno: cardinal;
err: cardinal;
eip: cardinal;
cs, __csh: word;
eflags: cardinal;
esp_at_signal: cardinal;
ss, __ssh: word;
fpstate: pfpstate;
oldmask: cardinal;
cr2: cardinal;
end;
```

The above records contain information about the processor state and process state at the moment a signal is sent to your program.

The records below are used in catching signals.

```
TSigAction = procedure(Sig: Longint; SigContext: SigContextRec);cdecl;
SignalHandler = Procedure ( Sig : Integer);cdecl;

PSignalHandler = SignalHandler;
SignalRestorer = Procedure;cdecl;
PSignalrestorer = SignalRestorer;
SigActionRec = packed record
  Handler : record
    case byte of
      0: (Sh: SignalHandler);
      1: (Sa: TSigAction);
    end;
  Sa_Mask : SigSet;
  Sa_Flags : Longint;
  Sa_restorer : SignalRestorer; { Obsolete - Don't use }
end;
PSigActionRec = ^SigActionRec;
```

Stat is used to store information about a file. It is defined in the syscalls unit.

```
stat = record
  dev : word;
  pad1 : word;
  ino : longint;
  mode : word;
  nlink : word;
```

```
uid      : word;
gid      : word;
rdev     : word;
pad2     : word;
size     : longint;
blksize  : Longint;
blocks   : Longint;
atime    : Longint;
unused1  : longint;
mtime    : Longint;
unused2  : longint;
ctime    : Longint;
unused3  : longint;
unused4  : longint;
unused5  : longint;
end;
```

Statfs is used to store information about a filesystem. It is defined in the syscalls unit.

```
statfs = record
  fstype   : longint;
  bsize    : longint;
  blocks   : longint;
  bfree    : longint;
  bavail   : longint;
  files    : longint;
  ffree    : longint;
  fsid     : longint;
  namelen  : longint;
  spare    : array [0..6] of longint;
end
```

Dir and PDir are used in the [OpenDir \(234\)](#) and [ReadDir \(236\)](#) functions.

```
TDir = record
  fd       : integer;
  loc      : longint;
  size     : integer;
  buf      : pdirent;
  nextoff  : longint;
  dd_max   : integer;
  lock     : pointer;
end;
PDir = ^TDir;
```

Dirent, PDirent are used in the [ReadDir \(236\)](#) function to return files in a directory.

```
PDirent = ^Dirent;
Dirent = Record
  ino,
  off   : longint;
  reclen : word;
  name   : string[255]
end;
```

Termio and Termios are used with `ioctl()` calls for terminal handling.

```
Const  NCCS = 19;
       NCC  = 8;
```

```
Type termio = record
  c_iflag,{ input mode flags }
  c_oflag,{ output mode flags }
  c_cflag,{ control mode flags }
  c_lflag : Word; { local mode flags }
  c_line  : Word; { line discipline - careful, only High byte in use}
  c_cc    : array [0..NCC-1] of char; { control characters }
end;
termios = record
  c_iflag,           { input mode flags }
  c_oflag,           { output mode flags }
  c_cflag,           { control mode flags }
  c_lflag : Cardinal; { local mode flags }
  c_line  : char;     { line discipline }
  c_cc    : array [0..NCCS-1] of char; { control characters }
end;
```

Utimbuf is used in the `Utime` ([252](#)) call to set access and modification time of a file.

```
utimbuf = record
  actime,modtime : Longint;
end;
```

For the `Select` ([240](#)) call, the following 4 types are needed:

```
FDSets = Array [0..31] of longint;
PFDSets = ^FDSets;
TimeVal = Record
  sec,usec : Longint;
end;
PTimeVal = ^TimeVal;
```

The `Uname` ([252](#)) function uses the `utsname` to return information about the current kernel :

```
utsname =record
  sysname,nodename,release,
  version,machine,domainname : Array[0..64] of char;
end;
```

Its elements are null-terminated C style strings, you cannot access them directly !

Variables

`Linuxerror` is the variable in which the procedures in the linux unit report errors.

```
LinuxError : Longint;
```

`StdErr` Is a Text variable, corresponding to Standard Error or diagnostic output. It is connected to file descriptor 2. It can be freely used, and will be closed on exit.

```
StdErr : Text;
```

Constants

Constants for setting/getting process priorities :

```
Prio_Process = 0;
Prio_PGrp    = 1;
Prio_User    = 2;
```

For testing access rights:

```
R_OK = 4;
W_OK = 2;
X_OK = 1;
F_OK = 0;
```

For signal handling functions :

```
SA_NOCLDSTOP = 1;
SA_SHIRQ     = $04000000;
SA_STACK     = $08000000;
SA_RESTART   = $10000000;
SA_INTERRUPT = $20000000;
SA_NOMASK    = $40000000;
SA_ONESHOT   = $80000000;
```

```
SIG_BLOCK = 0;
SIG_UNBLOCK = 1;
SIG_SETMASK = 2;
SIG_DFL = 0 ;
SIG_IGN = 1 ;
SIG_ERR = -1;
```

```
SIGHUP = 1;
SIGINT = 2;
SIGQUIT = 3;
SIGILL = 4;
SIGTRAP = 5;
SIGABRT = 6;
SIGIOT = 6;
SIGBUS = 7;
SIGFPE = 8;
SIGKILL = 9;
SIGUSR1 = 10;
SIGSEGV = 11;
SIGUSR2 = 12;
SIGPIPE = 13;
SIGALRM = 14;
SIGTERM = 15;
SIGSTKFLT = 16;
SIGCHLD = 17;
SIGCONT = 18;
SIGSTOP = 19;
SIGTSTP = 20;
SIGTTIN = 21;
SIGTTOU = 22;
```

```
SIGURG = 23;
SIGXCPU = 24;
SIGXFSZ = 25;
SIGVTALRM = 26;
SIGPROF = 27;
SIGWINCH = 28;
SIGIO = 29;
SIGPOLL = SIGIO;
SIGPWR = 30;
SIGUNUSED = 31;
```

For file control mechanism :

```
F_GetFd   = 1;
F_SetFd   = 2;
F_GetFl   = 3;
F_SetFl   = 4;
F_GetLk   = 5;
F_SetLk   = 6;
F_SetLkW  = 7;
F_GetOwn  = 8;
F_SetOwn  = 9;
```

For Terminal handling :

```
TCGETS = $5401 ;
TCSETS = $5402 ;
TCSETSW = $5403 ;
TCSETSF = $5404 ;
TCGETA = $5405 ;
TCSETA = $5406 ;
TCSETAW = $5407 ;
TCSETAF = $5408 ;
TCSBRK = $5409 ;
TCXONC = $540A ;
TCFLSH = $540B ;
TIOCEXCL = $540C ;
TIOCNXCL = $540D ;
TIOCSCTTY = $540E ;
TIOCGPGRP = $540F ;
TIOCSPGRP = $5410 ;
TIOCOUTQ = $5411 ;
TIOCSTI = $5412 ;
TIOCGWINSZ = $5413 ;
TIOCSWINSZ = $5414 ;
TIOCMGET = $5415 ;
TIOCMBIS = $5416 ;
TIOCMBIC = $5417 ;
TIOCMSET = $5418 ;
TIOCGSOFTCAR = $5419 ;
TIOCSSOFTCAR = $541A ;
FIONREAD = $541B ;
TIOCINQ = FIONREAD;
TIOCLINUX = $541C ;
```



```
TIOCCONS = $541D ;
TIOCGSERIAL = $541E ;
TIOCSSERIAL = $541F ;
TIOCPKT = $5420 ;
FIONBIO = $5421 ;
TIOCNOTTY = $5422 ;
TIOCSETD = $5423 ;
TIOCGETD = $5424 ;
TCSBRKP = $5425 ;
TIOCTTYGSTRUCT = $5426 ;
FIONCLEX = $5450 ;
FIOCLEX = $5451 ;
FIOASYNC = $5452 ;
TIOCSERCONFIG = $5453 ;
TIOCSERGWILD = $5454 ;
TIOCSERSWILD = $5455 ;
TIOCGLOCKTMIOS = $5456 ;
TIOCSLOCKTMIOS = $5457 ;
TIOCSERGSTRUCT = $5458 ;
TIOCSERGETLSR = $5459 ;
TIOCSERGETMULTI = $545A ;
TIOCSERSETMULTI = $545B ;
TIOCMWAIT = $545C ;
TIOCGICOUNT = $545D ;
TIOCPKT_DATA = 0;
TIOCPKT_FLUSHREAD = 1;
TIOCPKT_FLUSHWRITE = 2;
TIOCPKT_STOP = 4;
TIOCPKT_START = 8;
TIOCPKT_NOSTOP = 16;
TIOCPKT_DOSTOP = 32;
```

Other than that, all constants for setting the speed and control flags of a terminal line, as described in the `termios(2)` man page, are defined in the `linux` unit. It would take too much place to list them here. To check the mode field of a `stat` record, you can use the following constants :

```
{ Constants to check stat.mode }
STAT_IFMT = $f000; {00170000}
STAT_IFSOCK = $c000; {0140000}
STAT_IFLNK = $a000; {0120000}
STAT_IFREG = $8000; {0100000}
STAT_IFBLK = $6000; {0060000}
STAT_IFDIR = $4000; {0040000}
STAT_IFCHR = $2000; {0020000}
STAT_IFIFO = $1000; {0010000}
STAT_ISUID = $0800; {0004000}
STAT_ISGID = $0400; {0002000}
STAT_ISVTX = $0200; {0001000}
{ Constants to check permissions }
STAT_IRWXO = $7;
STAT_IROTH = $4;
STAT_IWOTH = $2;
STAT_IXOTH = $1;
STAT_IRWXG = STAT_IRWXO shl 3;
```

```
STAT_IRGRP = STAT_IROTH shl 3;
STAT_IWGRP = STAT_IWOTH shl 3;
STAT_IXGRP = STAT_IXOTH shl 3;
STAT_IRWXU = STAT_IRWXO shl 6;
STAT_IRUSR = STAT_IROTH shl 6;
STAT_IWUSR = STAT_IWOTH shl 6;
STAT_IXUSR = STAT_IXOTH shl 6;
```

You can test the type of a filesystem returned by a [FSStat \(213\)](#) call with the following constants:

```
fs_old_ext2 = $ef51;
fs_ext2     = $ef53;
fs_ext      = $137d;
fs_iso      = $9660;
fs_minix    = $137f;
fs_minix_30 = $138f;
fs_minix_V2 = $2468;
fs_msdos    = $4d44;
fs_nfs      = $6969;
fs_proc     = $9fa0;
fs_xia      = $012FD16D;
```

the [FLock \(211\)](#) call uses the following mode constants :

```
LOCK_SH = 1;
LOCK_EX = 2;
LOCK_UN = 8;
LOCK_NB = 4;
```

The [MMap \(231\)](#) function uses the following constants to specify access to mapped memory:

```
PROT_READ  = $1;    { page can be read }
PROT_WRITE = $2;    { page can be written }
PROT_EXEC  = $4;    { page can be executed }
PROT_NONE  = $0;    { page can not be accessed }
```

and the following constants to specify the type of mapping.

```
MAP_SHARED    = $1; { Share changes }
MAP_PRIVATE   = $2; { Changes are private }
MAP_TYPE      = $f; { Mask for type of mapping }
MAP_FIXED     = $10; { Interpret addr exactly }
MAP_ANONYMOUS = $20; { don't use a file }
```

12.2 Function list by category

What follows is a listing of the available functions, grouped by category. For each function there is a reference to the page where you can find the function.

File Input/Output routines

Functions for handling file input/output.

Name	Description	Page
Dup	Duplicate a file handle	199
Dup2	Copy one file handle to another	200
Fcntl	General file control	215
fdClose	Close file descriptor	207
fdFlush	Flush file descriptor	207
fdOpen	Open new file descriptor	207
fdRead	Read from file descriptor	208
fdSeek	Position in file	210
fdTruncate	Truncate file	210
fdWrite	Write to file descriptor	210
GetFS	Get file descriptor of pascal file	219
Select	Wait for input from file descriptor	240
SelectText	Wait for input from pascal file	241

General File handling routines

Functions for handling files on disk.

Name	Description	Page
Access	Check access rights on file	189
BaseName	Return name part of file	193
Chown	Change owner of file	194
Chmod	Change access rights on file	195
DirName	Return directory part of file	199
FSplit	Split filename in parts	212
FExpand	Return full-grown filename	210
FLock	Set lock on a file	211
FNMatch	Match filename to searchpattern	211
FSearch	Search for a file in a path	212
FSStat	Return filesystem information	213
FStat	Return file information	214
FRename	Rename file	216
LStat	Return information on a link	228
Link	Create a link	229
ReadLink	Read contents of a symbolic link	237
SymLink	Create a symbolic link	246
Umask	Set the file creation mask	251
UnLink	Remove a file	252
Utime	Change file timestamps	252

Pipes, FIFOs and streams

Functions for creating and managing pipes.

Name	Description	Page
AssignPipe	Create a pipe	190
AssignStream	Create pipes to program's input and output	191
MkFifo	Make a fifo	231
PClose	Close a pipe	235
POpen	Open a pipe for to program's input or output	236

Directory handling routines

Functions for reading and searching directories.

Name	Description	Page
CloseDir	Close directory handle	198
Glob	Return files matching a search expression	224
GlobFree	Free result of Glob	225
OpenDir	Open directory for reading	234
ReadDir	Read directory entry	236
SeekDir	Seek directory	239
TellDir	Seek directory	251

Process handling

Functions for managing processes and programs.

Name	Description	Page
Clone	Create a thread	196
Execl	Execute process with command-line list	201
Execle	Execute process with command-line list and environment	202
Execlp	Search in path and execute process with command list	203
Execv	Execute process	203
Execve	Execute process with environment	204
Execvp	Search in path and execute process	205
Fork	Spawn child process	216
GetEGid	Get effective group id	218
GetEnv	Get environment variable	219
GetEUid	Get effective user id	218
GetGid	Get group id	220
GetPid	Get process id	221
GetPPid	Get parent process id	221

GetPriority	Get process priority	222
GetUid	Get user id	224
Nice	Change priority of process	233
SetPriority	Change priority of process	241
Shell	Execute shell command	241
WaitPid	Wait for child process to terminate	253

Signals

Functions for managing and responding to signals.

Name	Description	Page
Alarm	Send alarm signal to self	190
Kill	Send arbitrary signal to process	228
pause	Wait for signal to arrive	235
SigAction	Set signal action	242
Signal	Set signal action	245
SigPending	See if signals are waiting	243
SigProcMask	Set signal processing mask	243
SigRaise	Send signal to self	244
SigSuspend	Sets signal mask and waits for signal	244

System information

Functions for retrieving system information such as date and time.

Name	Description	Page
GetDate	Return system date	217
GetDateTime	Return system date and time	217
GetDomainName	Return system domain name	217
GetEpochTime	Return epoch time	219
GetHostName	Return system host name	220
GetLocalTimezone	Return system timezone	221
GetTime	Return system time	222
GetTimeOfDay	Return system time	223
GetTimezoneFile	Return name of timezone file	223
ReadTimezoneFile	Read timezone file contents	239
SysInfo	Return general system information	247
Uname	Return system information	252

Terminal functions

Functions for controlling the terminal to which the process is connected.

Name	Description	Page
CFMakeRaw	Set terminal to raw mode	193
CFSetISpeed	Set terminal reading speed	193
CFSetOSpeed	Set terminal writing speed	194
IOCtl	General IO control call	225
IsATTY	See if filedescriptor is a terminal	226
TCDrain	Wait till all output was written	248
TCFlow	Suspend transmission or receipt of data	248
TCFlush	Discard data written to terminal	249
TCGetAttr	Get terminal attributes	249
TCGetPGrp	Return PID of foreground process	250
TCSendBreak	Send data for specific time	250
TCSetAttr	Set terminal attributes	250
TCSetPGrp	Set foreground process	251
TTYName	Name of tty file	251

Port input/output

Functions for reading and writing to the hardware ports.

Name	Description	Page
IOperm	Set permissions for port access	225
ReadPort	Read data from port	238
ReadPortB	Read 1 byte from port	238
ReadPortL	Read 4 bytes from port	238
ReadPortW	Read 2 bytes from port	239
WritePort	Write data to port	254
WritePortB	Write 1 byte to port	254
WritePortL	Write 4 bytes to port	254
WritePortW	Write 2 bytes to port	255

Utility routines

Auxiliary functions that are useful in connection with the other functions.

Name	Description	Page
CreateShellArgV	Create an array of pchars from string	198
EpochToLocal	Convert epoch time to local time	201
FD_Clr	Clear item of select filedescriptors	206

<code>FD_IsSet</code>	Check item of select filedescriptors	206
<code>FD_Set</code>	Set item of select filedescriptors	207
<code>FD_ZERO</code>	Clear all items in select filedescriptors	206
<code>LocalToEpoch</code>	Convert local time to epoch time	230
<code>MMap</code>	Map a file into memory	231
<code>MUnMap</code>	Unmap previously mapped memory file	233
<code>Octal</code>	Convert octal to digital	234
<code>S_ISBLK</code>	Check file mode for block device	226
<code>S_ISCHR</code>	Check file mode for character device	226
<code>S_ISDIR</code>	Check file mode for directory	226
<code>S_ISFIFO</code>	Check file mode for FIFO	226
<code>S_ISLNK</code>	Check file mode for symbolic link	227
<code>S_ISREG</code>	Check file mode for regular file	227
<code>S_ISSOCK</code>	Check file mode for socket	227
<code>StringToPPchar</code>	Create an array of pchars from string	245

12.3 Functions and procedures

Access

Declaration: `Function Access (Path : Pathstr; Mode : integer) : Boolean;`

Description: Tests user's access rights on the specified file. Mode is a mask existing of one or more of

R_OKUser has read rights.

W_OKUser has write rights.

X_OKUser has execute rights.

F_OKUser has search rights in the directory where the file is.

The test is done with the real user ID, instead of the effective user ID. If access is denied, or an error occurred, false is returned.

Errors: `LinuxError` is used to report errors:

sys_eaccessThe requested access is denied, either to the file or one of the directories in its path.

sys_einvalMode was incorrect.

sys_enoentA directory component in `Path` doesn't exist or is a dangling symbolic link.

sys_enotdirA directory component in `Path` is not a directory.

sys_enomemInsufficient kernel memory.

sys_eloop`Path` has a circular symbolic link.

See also: `Chown` (194), `Chmod` (195), `Access` (2)

Listing: `linuxex/ex26.pp`

Program Example26;

{ Program to demonstrate the Access function. }

Uses linux;

```
begin
  if Access ( '/etc/passwd',W_OK) then
    begin
      Writeln ( 'Better check your system. ');
      Writeln ( 'I can write to the /etc/passwd file ! ');
    end;
end.
```

Alarm

Declaration: Function Alarm(Sec : longint) : Longint;

Description: Alarm schedules an alarm signal to be delivered to your process in Sec seconds. When Sec seconds have elapsed, Linux will send a SIGALRM signal to the current process. If Sec is zero, then no new alarm will be set. Whatever the value of Sec, any previous alarm is cancelled.

The function returns the number of seconds till the previously scheduled alarm was due to be delivered, or zero if there was none.

Errors: None

Listing: linuxex/ex59.pp

Program Example59;

{ Program to demonstrate the Alarm function. }

Uses linux;

Procedure AlarmHandler(Sig : longint);cdecl;

```
begin
  Writeln ( 'Got to alarm handler ');
end;

begin
  Writeln ( 'Setting alarm handler ');
  Signal(SIGALRM, @AlarmHandler);
  Writeln ( 'Scheduling Alarm in 10 seconds ');
  Alarm(10);
  Writeln ( 'Pausing ');
  Pause;
  Writeln ( 'Pause returned ');
end.
```

AssignPipe

Declaration: Function AssignPipe(var pipe_in,pipe_out:longint):boolean; Function AssignPipe(var pipe_in,pipe_out:text):boolean; Function AssignPipe(var pipe_in,pipe_out:file):boolean;

Description: AssignPipe creates a pipe, i.e. two file objects, one for input, one for output. What is written to Pipe_out, can be read from Pipe_in.

This call is overloaded. The in and out pipe can take three forms: an typed or untyped file, a text file or a file descriptor.

If a text file is passed then reading and writing from/to the pipe can be done through the usual Readln(Pipe_in,...) and Writeln (Pipe_out,...) procedures.

The function returns True if everything went succesfully, False otherwise.

Errors: In case the function fails and returns False, LinuxError is used to report errors:

sys_enfileToo many file descriptors for this process.

sys_enfileThe system file table is full.

See also: POpen (236), MkFifo (231), pipe (2)

Listing: linuxex/ex36.pp

Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses linux;

Var pipi,pipo : Text;
s : String;

begin

```
Writeln ('Assigning Pipes. ');
If Not assignpipe(pipi,pipo) then
  Writeln('Error assigning pipes !',LinuxError);
Writeln ('Writing to pipe, and flushing. ');
Writeln (pipo,'This is a textstring');close(pipo);
Writeln ('Reading from pipe. ');
While not eof(pipi) do
  begin
    Readln (pipi,s);
    Writeln ('Read from pipe : ',s);
  end;
close (pipi);
writeln ('Closed pipes. ');
writeln
end.
```

AssignStream

Declaration: Function AssignStream(Var StreamIn,Streamout:text; Const Prog:String)
: longint; Function AssignStream(var StreamIn, StreamOut, StreamErr:
Text; const prog: String): LongInt;

Description: AssignStream creates a 2 or 3 pipes, i.e. two (or three) file objects, one for input, one for output,(and one for standard error) the other ends of these pipes are connected to standard input and output (and standard error) of Prog. Prog is the name of a program (including path) with options, which will be executed.

What is written to StreamOut, will go to the standard input of Prog. Whatever is written by Prog to it's standard output can be read from StreamIn. Whatever is written by Prog to it's standard error read from StreamErr, if present.

Reading and writing happens through the usual `Readln(StreamIn,...)` and `Writeln(StreamOut,...)` procedures.

Remark: You should *not* use `Reset` or `Rewrite` on a file opened with `POpen`. This will close the file before re-opening it again, thereby closing the connection with the program.

The function returns the process ID of the spawned process, or -1 in case of error.

Errors: In case of error (return value -1) `LinuxError` is used to report errors:

sys_enfile Too many file descriptors for this process.

sys_enfile The system file table is full.

Other errors include the ones by the `fork` and `exec` programs

See also: `AssignPipe` (190), `POpen` (236), `pipe` (2)

Listing: linuxex/ex38.pp

Program Example38;

{ Program to demonstrate the AssignStream function. }

Uses linux;

Var Si,So : Text;
S : **String**;
i : longint;

begin

if not (paramstr(1)='-son') **then**
 begin

Writeln ('Calling son');

 Assignstream (Si,So, './ex38 -son');

if linuxerror<>0 **then**

begin

writeln ('AssignStream failed !');

halt(1);

end;

Writeln ('Speaking to son');

For i:=1 **to** 10 **do**

begin

writeln (so,'Hello son !');

if ioresult<>0 **then** **writeln** ('Can''t speak to son...');

end;

For i:=1 **to** 3 **do** **writeln** (so,'Hello chap !');

 close (so);

while not eof(si) **do**

begin

readln (si,s);

writeln ('Father: Son said : ',S);

end;

Writeln ('Stopped conversation');

 Close (Si);

Writeln ('Put down phone');

end

Else

begin

Writeln ('This is the son');

While not eof (input) **do**

```
begin
  readln (s);
  if pos ('Hello son !',S)<>0 then
    Writeln ('Hello Dad !')
  else
    writeln ('Who are you ?');
  end;
  close (output);
end
end.
```

BaseName

Declaration: `Function BaseName (Const Path;Const Suf : Pathstr) : Pathstr;`

Description: Returns the filename part of Path, stripping off Suf if it exists. The filename part is the whole name if Path contains no slash, or the part of Path after the last slash. The last character of the result is not a slash, unless the directory is the root directory.

Errors: None.

See also: [DirName \(199\)](#), [FExpand \(210\)](#), [Basename \(1\)](#)

Listing: linuxex/ex48.pp

Program Example48;

{ Program to demonstrate the BaseName function. }

Uses linux;

Var S : **String**;

```
begin
  S:=FExpand(Paramstr(0));
  Writeln ('This program is called : ',Basename(S,''));
end.
```

CFMakeRaw

Declaration: `Procedure CFMakeRaw (var Tios:TermIOS);`

Description: CFMakeRaw Sets the flags in the Termios structure Tios to a state so that the terminal will function in Raw Mode.

Errors: None.

See also: [CFSetOSpeed \(194\)](#), [CFSetISpeed \(193\)](#), [termios \(2\)](#)

For an example, see [TCGetAttr \(249\)](#).

CFSetISpeed

Declaration: `Procedure CFSetISpeed (var Tios:TermIOS;Speed:Longint);`

Description: CFSetISpeed Sets the input baudrate in the TermIOS structure Tios to Speed.

Errors: None.

See also: CFSetOSpeed ([194](#)), CFMakeRaw ([193](#)), `termios` (2)

CFSetOSpeed

Declaration: `Procedure CFSetOSpeed (var Tios:TermIOS;Speed:Longint);`

Description: `CFSetOSpeed` Sets the output baudrate in the `Termios` structure `Tios` to `Speed`.

Errors: None.

See also: CFSetISpeed ([193](#)), CFMakeRaw ([193](#)), `termios` (2)

Chown

Declaration: `Function Chown (Path : Pathstr;NewUid,NewGid : Longint) : Boolean;`

Description: `Chown` sets the User ID and Group ID of the file in `Path` to `NewUid`, `NewGid`. The function returns `True` if the call was succesfull, `False` if the call failed.

Errors: Errors are returned in `LinuxError`.

sys_epermThe effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccessOne of the directories in `Path` has no search (=execute) permission.

sys_enoentA directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdirA directory entry in `OldPath` or `NewPath` is nor a directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe file is on a read-only filesystem.

sys_eloop`Path` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: `Chmod` ([195](#)), `Access` ([189](#)), `Chown` ((2))

Listing: `linuxex/ex24.pp`

Program `Example24`;

{ Program to demonstrate the Chown function. }

Uses `linux`;

Var `UID,GID : Longint;`
`F : Text;`

begin

```
WriteLn ('This will only work if you are root. ');
Write ('Enter a UID : ');readLn(UID);
Write ('Enter a GID : ');readLn(GID);
Assign (f,'test.txt');
Rewrite (f);
WriteLn (f,'The owner of this file should become : ');
```

```
Writeln ( f, 'UID : ', UID);
Writeln ( f, 'GID : ', GID);
Close (F);
if not Chown ( 'test.txt', UID, GID) then
  if LinuxError=Sys_EPERM then
    Writeln ( 'You are not root !')
  else
    Writeln ( 'Chmod failed with exit code : ', LinuxError)
else
  Writeln ( 'Changed owner successfully !');
end.
```

Chmod

Declaration: `Function Chmod (Path : Pathstr; NewMode : Longint) : Boolean;`

Description: Chmod Sets the Mode bits of the file in Path to NewMode. Newmode can be specified by 'or'-ing the following:

- S_ISUID**Set user ID on execution.
- S_ISGID**Set Group ID on execution.
- S_ISVTX**Set sticky bit.
- S_IRUSR**Read by owner.
- S_IWUSR**Write by owner.
- S_IXUSR**Execute by owner.
- S_IRGRP**Read by group.
- S_IWGRP**Write by group.
- S_IXGRP**Execute by group.
- S_IROTH**Read by others.
- S_IWOTH**Write by others.
- S_IXOTH**Execute by others.
- S_IRWXO**Read, write, execute by others.
- S_IRWXG**Read, write, execute by groups.
- S_IRWXU**Read, write, execute by user.

Errors: Errors are returned in `LinuxError`.

- sys_eperm**The effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.
- sys_eaccess**One of the directories in Path has no search (=execute) permission.
- sys_enoent**A directory entry in Path does not exist or is a symbolic link pointing to a non-existent directory.
- sys_enotdir**A directory entry in OldPath or NewPath is not a directory.
- sys_enomem**Insufficient kernel memory.
- sys_erofs**The file is on a read-only filesystem.
- sys_eloop**Path has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: Chown ([194](#)), Access ([189](#)), Chmod (`() 2`), Octal ([234](#))

Listing: linuxex/ex23.pp

Program Example23;

```
{ Program to demonstrate the Chmod function. }
```

Uses linux;

Var F : Text;

begin

```
  { Create a file }
  Assign (f, 'testex21');
  Rewrite (F);
  Writeln (f, '#!/bin/sh');
  Writeln (f, 'echo Some text for this file');
  Close (F);
  { Octal() makes the correct number from a
    number that LOOKS octal }
  Chmod ('testex21', octal (777));
  { File is now executable }
  execl ('./testex21');
```

end.

Clone

Declaration: TCloneFunc=function(args:pointer):longint;cdecl; Clone(func:TCloneFunc;sp:pointer;flags

Description: Clone creates a child process which is a copy of the parent process, just like Fork (216) does. In difference with Fork, however, the child process shares some parts of it's execution context with its parent, so it is suitable for the implementation of threads: many instances of a program that share the same memory.

When the child process is created, it starts executing the function Func, and passes it Args. The return value of Func is either the explicit return value of the function, or the exit code of the child process.

The sp pointer points to the memory reserved as stack space for the child process. This address should be the top of the memory block to be used as stack.

The Flags determine the behaviour of the Clone call. The low byte of the Flags contains the number of the signal that will be sent to the parent when the child dies. This may be bitwise OR'ed with the following constants:

CLONE_VMParent and child share the same memory space, including memory (un)mapped with subsequent mmap calls.

CLONE_FSParent and child have the same view of the filesystem; the chroot, chdir and umask calls affect both processes.

CLONE_FILESthe file descriptor table of parent and child is shared.

CLONE_SIGHANDthe parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PIDParent and child have the same process ID.

Clone returns the process ID in the parent process, and -1 if an error occurred.

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagainToo many processes are running.

`sys_enomem` Not enough memory to create child process.

See also: Fork ([216](#)), `clone` (2)

Listing: linuxex/ex71.pp

```
program TestC { lone };

uses
  Linux, Errors, crt;

const
  Ready : Boolean = false;
  aChar : Char    = 'a';

function CloneProc( Arg: Pointer ): LongInt; Cdecl;
begin
  WriteLn('Hello from the clone ', PChar(Arg));
  repeat
    Write(aChar);
    Select(0,0,0,0,600);
  until Ready;
  WriteLn('Clone finished. ');
  CloneProc := 1;
end;

var
  PID : LongInt;

procedure MainProc;
begin
  WriteLn('cloned process PID: ', PID );
  WriteLn('Press <ESC> to kill ... ');
  repeat
    Write(' ');
    Select(0,0,0,0,300);
    if KeyPressed then
      case ReadKey of
        #27: Ready := true;
        'a': aChar := 'A';
        'A': aChar := 'a';
        'b': aChar := 'b';
        'B': aChar := 'B';
      end;
    until Ready;
  WriteLn('Ready. ');
end;

const
  StackSize = 16384;
  theFlags = CLONE_VM+CLONE_FS+CLONE_FILES+CLONE_SIGHAND;
  aMsg      : PChar = 'Oops !';

var
  theStack : Pointer;
  ExitStat : LongInt;

begin
  GetMem(theStack, StackSize);
```

```
PID := Clone( @CloneProc,
              Pointer( LongInt(theStack)+StackSize),
              theFlags,
              aMsg);
if PID < 0 then
  WriteLn('Error : ', LinuxError, ' when cloning.')
else
  begin
    MainProc;
    case WaitPID(0, @ExitStat, Wait_Untraced or wait_clone) of
      -1: WriteLn('error:', LinuxError, '; ', StrError(LinuxError));
      0: WriteLn('error:', LinuxError, '; ', StrError(LinuxError));
    else
      WriteLn('Clone exited with: ', ExitStat shr 8);
    end;
  end;
  FreeMem( theStack, StackSize );
end.
```

CloseDir

Declaration: `Function CloseDir (p:ppdir) : integer;`

Description: `CloseDir` closes the directory pointed to by `p`. It returns zero if the directory was closed successfully, -1 otherwise.

Errors: Errors are returned in `LinuxError`.

See also: `OpenDir` (234), `ReadDir` (236), `SeekDir` (239), `TellDir` (251), `closedir` (3)

For an example, see `OpenDir` (234).

CreateShellArgV

Declaration: `function CreateShellArgV(const prog:string):ppchar; function CreateShellArgV(const prog:Ansistring):ppchar;`

Description: `CreateShellArgV` creates an array of 3 `PChar` pointers that can be used as arguments to `ExecVE` the first elements in the array will contain `/bin/sh`, the second will contain `-c`, and the third will contain `prog`.

The function returns a pointer to this array, of type `PPChar`.

Errors: None.

See also: `Shell` (241)

Listing: `linuxex/ex61.pp`

Program `ex61;`

{ Example program to demonstrate the CreateShellArgV function }

uses `linux;`

Var

`S: String;`


```

PP : PPchar;
I : longint;

begin
  S:= 'script -a -b -c -d -e fghijk';
  PP:=CreateShellArgV(S);
  I:=0;
  If PP<>Nil then
    While PP[I]<>Nil do
      begin
        Writeln ( 'Got : "',PP[I], '"' );
        Inc(I);
      end;
    end.

```

DirName

Declaration: Function DirName (Const Path : Pathstr) : Pathstr;

Description: Returns the directory part of Path. The directory is the part of Path before the last slash, or empty if there is no slash. The last character of the result is not a slash, unless the directory is the root directory.

Errors: None.

See also: BaseName ([193](#)), FExpand ([210](#)), Dirname (1)

Listing: linuxex/ex47.pp

Program Example47;

{ Program to demonstrate the DirName function. }

Uses linux;

Var S : **String**;

```

begin
  S:=FExpand(Paramstr(0));
  Writeln ( 'This program is in directory : ',Dirname(S));
end.

```

Dup

Declaration: Function Dup(oldfile:longint;var newfile:longint):Boolean; Function Dup(var oldfile,newfile:text):Boolean; Function Dup(var oldfile,newfile:file):Boolean;

Description: Makes NewFile an exact copy of OldFile, after having flushed the buffer of OldFile in case it is a Text file or untyped file. Due to the buffering mechanism of Pascal, this has not the same functionality as the dup (2) call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns False in case of an error, True if successful.

Errors: In case of errors, Linuxerror is used to report errors.

sys_ebadfOldFile hasn't been assigned.

sys_emfileMaximum number of open files for the process is reached.

See also: Dup2 ([200](#)), Dup (2)

Listing: linuxex/ex31.pp

```
program Example31;  
  
  { Program to demonstrate the Dup function. }  
  
uses linux;  
  
var f : text;  
  
begin  
  if not dup (output,f) then  
    writeln ('Dup Failed !');  
    writeln ('This is written to stdout.');
```

Dup2

Declaration: Function Dup2(oldfile,newfile:longint):Boolean; Function Dup2(var oldfile,newfile:text)
Function Dup2(var oldfile,newfile:file):Boolean;

Description: Makes NewFile an exact copy of OldFile, after having flushed the buffer of OldFile in the case of text or untyped files.

NewFile can be an assigned file. If newfile was open, it is closed first. Due to the buffering mechanism of Pascal, this has not the same functionality as the dup2 (2) call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns True if succesful, false otherwise.

Errors: In case of error, Linuxerror is used to report errors.

sys_ebadfOldFile hasn't been assigned.

sys_emfileMaximum number of open files for the process is reached.

See also: Dup ([199](#)), Dup2 (2)

Listing: linuxex/ex32.pp

```
program Example31;  
  
  { Program to demonstrate the Dup function. }  
  
uses linux;  
  
var f : text;  
    i : longint;  
  
begin
```

```

Assign (f, 'text.txt');
Rewrite (F);
For i:=1 to 10 do writeln (F, 'Line : ', i);
if not dup2 (output, f) then
  Writeln ('Dup2 Failed !');
writeln ('This is written to stdout. ');
writeln (f, 'This is written to the dup file, and flushed');
flush(f);
writeln;
{ Remove file. Comment this if you want to check flushing. }
Unlink ('text.txt');
end.

```

EpochToLocal

Declaration: Procedure EpochToLocal (Epoch : Longint; var Year, Month, Day, Hour, Minute, Second : Word);

Description: Converts the epoch time (=Number of seconds since 00:00:00 , January 1, 1970, corrected for your time zone) to local date and time.

This function takes into account the timzeone settings of your system.

Errors: None

See also: [GetEpochTime \(219\)](#), [LocalToEpoch \(230\)](#), [GetTime \(222\)](#), [GetDate \(217\)](#)

Listing: linuxex/ex3.pp

Program Example3;

{ Program to demonstrate the EpochToLocal function . }

Uses linux;

Var Year, month, day, hour, minute, seconds : Word;

begin

EpochToLocal (GetEpochTime, Year, month, day, hour, minute, seconds);

Writeln (' Current date : ', Day:2, '/', Month:2, '/', Year:4);

Writeln (' Current time : ', Hour:2, ':', minute:2, ':', seconds:2);

end.

Execl

Declaration: Procedure Execl (Path : pathstr);

Description: Replaces the currently running program with the program, specified in path. Path is split into a command and it's options. The executable in path is NOT searched in the path. The current environment is passed to the program. On success, execl does not return.

Errors: Errors are reported in `LinuxError`:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve ([204](#)), Execv ([203](#)), Execvp ([205](#)), Execle ([202](#)), Execlp ([203](#)), Fork ([216](#)), execvp (3)

Listing: linuxex/ex10.pp

Program Example10;

{ Program to demonstrate the Execl function. }

Uses linux , strings ;

begin

{ Execute 'ls -l', with current environment. }
{ 'ls' is NOT looked for in PATH environment variable. }
Execl ('/bin/ls -l');

end.

Execle

Declaration: Procedure Execle (Path : pathstr, Ep : ppchar);

Description: Replaces the currently running program with the program, specified in path. Path is split into a command and it's options. The executable in path is searched in the path, if it isn't an absolute filename. The environment in ep is passed to the program. On success, execle does not return.

Errors: Errors are reported in LinuxError:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve ([204](#)), Execv ([203](#)), Execvp ([205](#)), Execl ([201](#)), Execlp ([203](#)), Fork ([216](#)), execvp (3)

Listing: linuxex/ex11.pp

Program Example11;

{ Program to demonstrate the Execle function. }

Uses linux , strings ;

```
begin
  { Execute 'ls -l', with current environment. }
  { 'ls' is NOT looked for in PATH environment variable. }
  { envp is defined in the system unit. }
  Execl ( '/bin/ls -l', envp );
end.
```

Execlp

Declaration: Procedure Execlp (Path : pathstr);

Description: Replaces the currently running program with the program, specified in path. Path is split into a command and it's options. The executable in path is searched in the path, if it isn't an absolute filename. The current environment is passed to the program. On success, execlp does not return.

Errors: Errors are reported in LinuxError:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve (204), Execv (203), Execvp (205), Execl (202), Execl (201), Fork (216), execvp (3)

Listing: linuxex/ex12.pp

Program Example12;

```
{ Program to demonstrate the Execlp function. }
```

Uses linux, strings;

```
begin
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable. }
  { envp is defined in the system unit. }
  Execlp ( 'ls -l', envp );
end.
```

Execv

Declaration: Procedure Execv (Path : pathstr; args : ppchar);

Description: Replaces the currently running program with the program, specified in path. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, execv does not return.

Errors: Errors are reported in LinuxError:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve (204), Execvp (205), Execle (202), Execl (201), Execlp (203), Fork (216), execv (3)

Listing: linux/ex8.pp

Program Example8;

{ Program to demonstrate the Execv function. }

Uses linux , strings ;

Const Arg0 : PChar = '/bin/lS';
Arg1 : Pchar = '-l';

Var PP : PPchar;

begin

GetMem (PP,3*SizeOf(Pchar));
PP[0]:=Arg0;
PP[1]:=Arg1;
PP[3]:=Nil;
{ Execute '/bin/lS -l', with current environment }
Execv ('/bin/lS',pp);

end.

Execve

Declaration: Procedure Execve(Path:pchar;args:ppchar;ep:ppchar); Procedure Execve
(Path : pathstr; args,ep : ppchar);

Description: Replaces the currently running program with the program, specified in path. It gives the program the options in args, and the environment in ep. They are pointers to an array of pointers to null-terminated strings. The last pointer in this array should be nil. On success, execve does not return.

Errors: Errors are reported in LinuxError:

eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve (204), Execv (203), Execvp (205) Execle (202), Execl (201), Execlp (203), Fork (216), execve (2)

Listing: linuxex/ex7.pp

Program Example7;

{ Program to demonstrate the Execve function. }

Uses linux , strings ;

Const Arg0 : PChar = '/bin/l's' ;
 Arg1 : Pchar = '-l' ;

Var PP : PPchar ;

begin

GetMem (PP,3***SizeOf**(Pchar));
 PP[0]:=Arg0;
 PP[1]:=Arg1;
 PP[3]:=Nil;
 { Execute '/bin/l's -l', with current environment }
 { Env is defined in system.inc }
 ExecVe ('/bin/l's',pp,envp);

end.

Execvp

Declaration: Procedure Execvp (Path : pathstr; args : ppchar);

Description: Replaces the currently running program with the program, specified in path. The executable in path is searched in the path, if it isn't an absolute filename. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, execvp does not return.

Errors: Errors are reported in LinuxError:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve (204), Execv (203), Execle (202), Execl (201), Execlp (203), Fork (216), execvp (3)

Listing: linuxex/ex9.pp

Program Example9;

{ Program to demonstrate the Execvp function. }

Uses linux, strings;

Const Arg0 : PChar = 'ls';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

begin

GetMem (PP, 3***SizeOf**(Pchar));
 PP[0] := Arg0;
 PP[1] := Arg1;
 PP[3] := Nil;
 { Execute 'ls -l', with current environment. }
 { 'ls' is looked for in PATH environment variable. }
 { Env is defined in the system unit. }
 Execvp ('ls', pp, envp);

end.

FD_ZERO

Declaration: Procedure FD_ZERO (var fds:fdSet);

Description: FD_ZERO clears all the filedescriptors in the file descriptor set fds.

Errors: None.

See also: Select ([240](#)), SelectText ([241](#)), GetFS ([219](#)), FD_Clr ([206](#)), FD_Set ([207](#)), FD_IsSet ([206](#))

For an example, see Select ([240](#)).

FD_Clr

Declaration: Procedure FD_Clr (fd:longint;var fds:fdSet);

Description: FD_Clr clears file descriptor fd in filedescriptor set fds.

Errors: None.

See also: Select ([240](#)), SelectText ([241](#)), GetFS ([219](#)), FD_ZERO ([206](#)), FD_Set ([207](#)), FD_IsSet ([206](#))

For an example, see Select ([240](#)).

FD_IsSet

Declaration: Function FD_IsSet (fd:longint;var fds:fdSet) : boolean;

Description: FD_Set Checks whether file descriptor fd in filedescriptor set fds is set.

Errors: None.

See also: [Select \(240\)](#), [SelectText \(241\)](#), [GetFS \(219\)](#), [FD_ZERO \(206\)](#), [FD_Clr \(206\)](#), [FD_Set \(207\)](#)

For an example, see [Select \(240\)](#).

FD_Set

Declaration: `Procedure FD_Set (fd:longint;var fds:fdSet);`

Description: `FD_Set` sets file descriptor `fd` in filedescriptor set `fds`.

Errors: None.

See also: [Select \(240\)](#), [SelectText \(241\)](#), [GetFS \(219\)](#), [FD_ZERO \(206\)](#), [FD_Clr \(206\)](#), [FD_IsSet \(206\)](#)

For an example, see [Select \(240\)](#).

fdClose

Declaration: `Function fdClose (fd:longint) : boolean;`

Description: `fdClose` closes a file with file descriptor `Fd`. The function returns `True` if the file was closed successfully, `False` otherwise.

Errors: Errors are returned in `LinuxError`

See also: [fdOpen \(207\)](#), [fdRead \(208\)](#), [fdWrite \(210\)](#), [fdTruncate \(210\)](#), [fdFlush \(207\)](#), [seefdSeek](#)

For an example, see [fdOpen \(207\)](#).

fdFlush

Declaration: `Function fdFlush (fd:Longint) : boolean;`

Description: `fdflush` flushes the Linux kernel file buffer, so the file is actually written to disk. This is NOT the same as the internal buffer, maintained by Free Pascal. The function returns `True` if the call was successful, `false` if an error occurred.

Errors: Errors are returned in `LinuxError`.

See also: [fdOpen \(207\)](#), [fdClose \(207\)](#), [fdRead \(208\)](#), [fdWrite \(210\)](#), [fdTruncate \(210\)](#), [fdSeek \(210\)](#)

For an example, see [fdRead \(208\)](#).

fdOpen

Declaration: `Function fdOpen(PathName:String;flags:longint):longint; Function fdOpen(PathName:Pchar ;flags:longint):longint; Function fdOpen(PathName:String;flags,mode:longint):longint; Function fdOpen(PathName:Pchar ;flags,mode:longint):longint;`

Description: `fdOpen` opens a file in `PathName` with flags `flags` One of the following:

Open_RdOnlyFile is opened Read-only.

Open_WrOnlyFile is opened Write-only.

Open_RdWrFile is opened Read-Write.

The flags may be OR-ed with one of the following constants:

Open_AccmodeFile is opened

Open_CreatFile is created if it doesn't exist.

Open_ExclIf the file is opened with `Open_Creat` and it already exists, the call will fail.

Open_NoCttyIf the file is a terminal device, it will NOT become the process' controlling terminal.

Open_TruncIf the file exists, it will be truncated.

Open_Appendthe file is opened in append mode. *Before each write*, the file pointer is positioned at the end of the file.

Open_NonBlockThe file is opened in non-blocking mode. No operation on the file descriptor will cause the calling process to wait till.

Open_NDelayIdem as `Open_NonBlock`

Open_SyncThe file is opened for synchronous IO. Any write operation on the file will not return until the data is physically written to disk.

Open_NoFollowif the file is a symbolic link, the open fails. (LINUX 2.1.126 and higher only)

Open_Directoryif the file is not a directory, the open fails. (LINUX 2.1.126 and higher only)

`PathName` can be of type `PChar` or `String`. The optional mode argument specifies the permissions to set when opening the file. This is modified by the `umask` setting. The real permissions are `Mode` and not `umask`. The return value of the function is the file descriptor, or a negative value if there was an error.

Errors: Errors are returned in `LinuxError`

See also: `fdClose` (207), `fdRead` (208), `fdWrite` (210), `fdTruncate` (210), `fdFlush` (207), `fdSeek` (210)

Listing: linuxex/ex19.pp

Program Example19;

```
{ Program to demonstrate the fdOpen, fdwrite and fdClose functions. }
```

Uses linux;

Const Line : **String**[80] = 'This is easy writing !';

Var FD : Longint;

begin

```
FD:=fdOpen ( 'Test.dat', Open_WrOnly or Open_Creat);
```

```
if FD>0 then
```

```
begin
```

```
if length(Line)<>fdwrite (FD,Line[1],Length(Line)) then
```

```
Writeln ( 'Error when writing to file !');
```

```
fdClose(FD);
```

```
end;
```

```
end.
```

fdRead

Declaration: `Function fdRead (fd:longint;var buf;size:longint) : longint;`

Description: `fdRead` reads at most `size` bytes from the file descriptor `fd`, and stores them in `buf`. The function returns the number of bytes actually read, or -1 if an error occurred. No checking on the length of `buf` is done.

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (207), `fdClose` (207), `fdWrite` (210), `fdTruncate` (210), `fdFlush` (207), `fdSeek` (210)

Listing: `linuxex/ex20.pp`

Program `Example20`;

{ Program to demonstrate the fdRead and fdTruncate functions. }

Uses `linux`;

Const `Data : string[10] = '12345687890'`;

Var `FD : Longint`;
 `I : longint`;

begin

`FD:=fdOpen('test.dat',open_wronly or open_creat,octal(666));`

if `fd>0` **then**

begin

{ Fill file with data }

for `I:=1 to 10` **do**

if `fdWrite (FD,Data[1],10)<>10` **then**

begin

`writeln ('Error when writing !');`

`halt(1);`

end;

`fdClose(FD);`

`FD:=fdOpen('test.dat',open_rdonly);`

{ Read data again }

if `FD>0` **then**

begin

for `I:=1 to 5` **do**

if `fdRead (FD,Data[1],10)<>10` **then**

begin

`Writeln ('Error when Reading !');`

`Halt(2);`

end;

`fdClose(FD);`

{ Truncating file at 60 bytes }

{ For truncating, file must be open or write }

`FD:=fdOpen('test.dat',open_wronly,octal(666));`

if `FD>0` **then**

begin

if not `fdTruncate(FD,60)` **then**

`Writeln('Error when truncating !');`

`fdClose (FD);`

end;

end;

end;

end.

fdSeek

Declaration: `Function fdSeek (fd,Pos,SeekType:longint) : longint;`

Description: `fdSeek` sets the current fileposition of file `fd` to `Pos`, starting from `SeekType`, which can be one of the following:

Seek_Set `Pos` is the absolute position in the file.

Seek_Cur `Pos` is relative to the current position.

Seek_end `Pos` is relative to the end of the file.

The function returns the new fileposition, or -1 if an error occurred.

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (207), `fdWrite` (210), `fdClose` (207), `fdRead` (208), `fdTruncate` (210), `fdFlush` (207)

For an example, see `fdOpen` (207).

fdTruncate

Declaration: `Function fdTruncate (fd,size:longint) : boolean;`

Description: `fdTruncate` sets the length of a file in `fd` on `size` bytes, where `size` must be less than or equal to the current length of the file in `fd`. The function returns `True` if the call was successful, `false` if an error occurred.

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (207), `fdClose` (207), `fdRead` (208), `fdWrite` (210), `fdFlush` (207), `fdSeek` (210)

fdWrite

Declaration: `Function fdWrite (fd:longint;var buf;size:longint) : longint;`

Description: `fdWrite` writes at most `size` bytes from `buf` to file descriptor `fd`. The function returns the number of bytes actually written, or -1 if an error occurred.

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (207), `fdClose` (207), `fdRead` (208), `fdTruncate` (210), `fdSeek` (210), `fdFlush` (207)

FExpand

Declaration: `Function FExpand (Const Path: Pathstr) : pathstr;`

Description: Expands `Path` to a full path, starting from root, eliminating directory references such as `.` and `..` from the result.

Errors: None

See also: `BaseName` (193), `DirName` (199)

Listing: `linuxex/ex45.pp`

Program Example45;

{ Program to demonstrate the FExpand function. }

Uses linux;

begin

WriteLn ('This program is in : ',FExpand(**Paramstr**(0)));
end.

FLock

Declaration: Function Flock (fd,mode : longint) : boolean; Function Flock (var T : text;mode : longint) : boolean; Function Flock (var F : File;mode : longint) : boolean;

Description: FLock implements file locking. it sets or removes a lock on the file F. F can be of type Text or File, or it can be a LINUX filedescriptor (a longint) Mode can be one of the following constants :

LOCK_SH sets a shared lock.

LOCK_EX sets an exclusive lock.

LOCK_UN unlocks the file.

LOCK_NB This can be OR-ed together with the other. If this is done the application doesn't block when locking.

The function returns True if successful, False otherwise.

Errors: If an error occurs, it is reported in `LinuxError`.

See also: `Fcntl` ([215](#)), `flock` (2)

FNMatch

Declaration: Function FNMatch(const Pattern,Name:string):Boolean;

Description: FNMatch returns True if the filename in Name matches the wildcard pattern in Pattern, False otherwise.

Pattern can contain the wildcards * (match zero or more arbitrary characters) or ? (match a single character).

Errors: None.

See also: `FSearch` ([212](#)), `FExpand` ([210](#))

Listing: linuxex/ex69.pp

Program Example69;

{ Program to demonstrate the FNMatch function. }

Uses linux;

Procedure TestMatch(Pattern,Name : String);

begin

```
    Write ( '""',Name, '"" ');
    If FNMatch ( Pattern,Name) then
        Write ( 'matches' )
    else
        Write ( 'does not match' );
    Writeln ( ' ""', Pattern, '"".' );
end;

begin
    TestMatch ( '*' , ' FileName' );
    TestMatch ( '. * ' , ' FileName' );
    TestMatch ( '* a * ' , ' FileName' );
    TestMatch ( '? ile * ' , ' FileName' );
    TestMatch ( '? ' , ' FileName' );
    TestMatch ( '. ? ' , ' FileName' );
    TestMatch ( '? a * ' , ' FileName' );
    TestMatch ( '? ? * me ? ' , ' FileName' );
end.
```

FSearch

Declaration: `Function FSearch (Path : pathstr;DirList : string) : Pathstr;`

Description: Searches in DirList, a colon separated list of directories, for a file named Path. It then returns a path to the found file.

Errors: An empty string if no such file was found.

See also: [BaseName \(193\)](#), [DirName \(199\)](#), [FExpand \(210\)](#), [FNMatch \(211\)](#)

Listing: linuxex/ex46.pp

Program Example46;

{ Program to demonstrate the FSearch function. }

Uses linux, strings;

begin

Writeln ('ls is in : ',FSearch ('ls',strpas (Getenv ('PATH'))));

end.

FSplit

Declaration: `Procedure FSplit(const Path:PathStr;
 Var Dir:DirStr;Var Name:NameStr;Var Ext:ExtStr);`

Description: FSplit splits a full file name into 3 parts : A Path, a Name and an extension (in ext). The extension is taken to be all letters after the last dot (.).

Errors: None.

See also: [FSearch \(212\)](#)

Listing: linuxex/ex67.pp

```
Program Example67;

uses Linux;

{ Program to demonstrate the FSplit function. }

var
  Path,Name,Ext : string;

begin
  FSplit(ParamStr(1),Path,Name,Ext);
  WriteLn(' Split ',ParamStr(1),' in:');
  WriteLn(' Path      : ',Path);
  WriteLn(' Name      : ',Name);
  WriteLn(' Extension : ',Ext);
end.
```

FSStat

Declaration: Function FSStat (Path : Pathstr; Var Info : statfs) : Boolean; Function FSStat (Fd:longint;Var Info:stat) : Boolean;

Description: Return in Info information about the filesystem on which the file Path resides, or on which the file with file descriptor fd resides. Info is of type statfs. The function returns True if the call was succesfull, False if the call failed.

Errors: LinuxError is used to report errors.

sys_enotdirA component of Path is not a directory.
sys_einvalInvalid character in Path.
sys_enoentPath does not exist.
sys_eaccessSearch permission is denied for component in Path.
sys_eloopA circular symbolic link was encountered in Path.
sys_eioAn error occurred while reading from the filesystem.

See also: FStat ([214](#)), LStat ([228](#)), statfs (2)

Listing: linuxex/ex30.pp

```
program Example30;

{ Program to demonstrate the FSStat function. }

uses linux;

var s : string;
    info : statfs;

begin
  writeln ('Info about current partition : ');
  s:= '.';
  while s<>'q' do
    begin
      if not fsstat (s,info) then
        begin
```

```
        writeln('Fstat failed. Errno : ',linuxerror);
        halt (1);
    end;
writeln;
writeln ('Result of fsstat on file ',s,'.');
writeln ('fstype   : ',info.fstype);
writeln ('bsize    : ',info.bsize);
writeln ('bfree     : ',info.bfree);
writeln ('bavail    : ',info.bavail);
writeln ('files     : ',info.files);
writeln ('ffree     : ',info.ffree);
writeln ('fsid      : ',info.fsid);
writeln ('Namelen   : ',info.namelen);
write ('Type name of file to do fsstat. (q quits) : ');
readln (s)
end;
end.
```

FStat

Declaration: `Function FStat(Path:Pathstr;Var Info:stat):Boolean;` `Function FStat(Fd:longint;Var Info:stat):Boolean;` `Function FStat(var F:Text;Var Info:stat):Boolean;` `Function FStat(var F:File;Var Info:stat):Boolean;`

Description: FStat gets information about the file specified in one of the following:

Patha file on the filesystem.

Fda valid file descriptor.

Fan opened text file or untyped file.

and stores it in **Info**, which is of type `stat`. The function returns `True` if the call was succesfull, `False` if the call failed.

Errors: `LinuxError` is used to report errors.

`sys_enoent`Path does not exist.

See also: `FSSStat` ([213](#)), `LStat` ([228](#)), `stat` ([2](#))

Listing: linuxex/ex28.pp

```
program example28;

{ Program to demonstrate the FStat function. }

uses linux;

var f : text;
    i : byte;
    info : stat;

begin
    { Make a file }
    assign (f,'test.fil');
    rewrite (f);
    for i:=1 to 10 do writeln (f,'Testline # ',i);
    close (f);
```



```
{ Do the call on made file . }
if not fstat ('test.fil',info) then
begin
  writeln('Fstat failed. Errno : ',linuxerror);
  halt (1);
end;
writeln;
writeln ('Result of fstat on file ''test.fil''.');
writeln ('Inode   : ',info.ino);
writeln ('Mode    : ',info.mode);
writeln ('nlink   : ',info.nlink);
writeln ('uid     : ',info.uid);
writeln ('gid     : ',info.gid);
writeln ('rdev    : ',info.rdev);
writeln ('Size    : ',info.size);
writeln ('Blksize  : ',info.blksize);
writeln ('Blocks  : ',info.blocks);
writeln ('atime   : ',info.atime);
writeln ('mtime   : ',info.mtime);
writeln ('ctime   : ',info.ctime);
{ Remove file }
erase (f);
end.
```

Fcntl

Declaration: `Function Fcntl(Fd:longint;Cmd:Integer):integer; Function Fcntl(var Fd:Text;Cmd:Integer)`

Description: Read a file's attributes. `Fd` is an assigned file, or a valid file descriptor. `Cmd` specifies what to do, and is one of the following:

F_GetFdRead the `close_on_exec` flag. If the low-order bit is 0, then the file will remain open across `execve` calls.

F_GetFlRead the descriptor's flags.

F_GetOwnGet the Process ID of the owner of a socket.

Errors: `LinuxError` is used to report errors.

sys_ebadf`Fd` has a bad file descriptor.

See also: [Fcntl \(215\)](#), [Fcntl \(2\)](#)

Fcntl

Declaration: `Procedure Fcntl (Fd : text, Cmd : Integer; Arg : longint); Procedure Fcntl (Fd:longint;Cmd:longint;Arg:Longint);`

Description: Read or Set a file's attributes. `Fd` is an assigned file or a valid file descriptor. `Cmd` specifies what to do, and is one of the following:

F_SetFdSet the `close_on_exec` flag of `Fd`. (only the least significant bit is used).

F_GetLkReturn the `flock` record that prevents this process from obtaining the lock, or set the `l_type` field of the lock of there is no obstruction. `Arg` is a pointer to a `flock` record.

F_SetLkSet the lock or clear it (depending on `l_type` in the `flock` structure). if the lock is held by another process, an error occurs.

F_GetLkw Same as for **F_Setlk**, but wait until the lock is released.

F_SetOwn Set the Process or process group that owns a socket.

Errors: `LinuxError` is used to report errors.

sys_ebadf `Fd` has a bad file descriptor.

sys_eagain or **sys_eaccess** For **F_SetLk**, if the lock is held by another process.

See also: `Fcntl` ([215](#)), `Fcntl` (2), `seefLock`

Fork

Declaration: `Function Fork : Longint;`

Description: `Fork` creates a child process which is a copy of the parent process. `Fork` returns the process ID in the parent process, and zero in the child's process. (you can get the parent's PID with `GetPPid` ([221](#))).

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Not enough memory to create child process.

See also: `Execve` ([204](#)), `Clone` ([196](#)), `fork` (2)

FRename

Declaration: `Function FReName (OldName, NewName : Pchar) : Boolean;` `Function FReName (OldName, NewName : String) : Boolean;`

Description: `FRename` renames the file `OldName` to `NewName`. `NewName` can be in a different directory than `OldName`, but it cannot be on another partition (device). Any existing file on the new location will be replaced.

If the operation fails, then the `OldName` file will be preserved.

The function returns `True` on succes, `False` on failure.

Errors: On error, errors are reported in `LinuxError`. Possible errors include:

sys_eisdir `NewName` exists and is a directory, but `OldName` is not a directory.

sys_exdev `NewName` and `OldName` are on different devices.

sys_enotempty or **sys_eexist** `NewName` is an existing, non-empty directory.

sys_ebusy `OldName` or `NewName` is a directory and is in use by another process.

sys_einval `NewName` is part of `OldName`.

sys_mlink `OldPath` or `NewPath` already have the maximum amount of links pointing to them.

sys_enotdir part of `OldName` or `NewName` is not directory.

sys_efault For the `pchar` case: One of the pointers points to an invalid address.

sys_eaccess access is denied when attempting to move the file.

sys_enametoolong Either `OldName` or `NewName` is too long.

sys_enoent directory component in `OldName` or `NewName` didn't exist.

sys_enomem not enough kernel memory.

sys_erofs `NewName` or `OldName` is on a read-only file system.

sys_eloop too many symbolic links were encountered trying to expand `OldName` or `NewName`

sys_enospc the filesystem has no room for the new directory entry.

See also: `UnLink` ([252](#))

GetDate

Declaration: `Procedure GetDate (Var Year, Month, Day : Word) ;`

Description: Returns the current date.

Errors: None

See also: [GetEpochTime \(219\)](#), [GetTime \(222\)](#), [GetDateTime \(217\)](#), [EpochToLocal \(201\)](#)

Listing: linuxex/ex6.pp

Program Example6;

{ Program to demonstrate the GetDate function. }

Uses linux;

Var Year, Month, Day : Word;

begin

 GetDate (Year, Month, Day);

WriteLn ('Date : ', Day:2, '/', Month:2, '/', Year:4);

end.

GetDateTime

Declaration: `Procedure GetDateTime(Var Year,Month,Day,hour,minute,second:Word) ;`

Description: Returns the current date and time. The time is corrected for the local time zone. This procedure is equivalent to the [GetDate \(217\)](#) and [GetTime](#) calls.

Errors: None

See also: [GetEpochTime \(219\)](#), [GetTime \(222\)](#), [EpochToLocal \(201\)](#), [GetDate \(217\)](#)

Listing: linuxex/ex60.pp

Program Example6;

{ Program to demonstrate the GetDateTime function. }

Uses linux;

Var Year, Month, Day, Hour, min, sec : Word;

begin

 GetDateTime (Year, Month, Day, Hour, min, sec);

WriteLn ('Date : ', Day:2, '/', Month:2, '/', Year:4);

WriteLn ('Time : ', Hour:2, ':', Min:2, ':', Sec:2);

end.

GetDomainName

Declaration: `Function GetDomainName : String;`

Description: Get the domain name of the machine on which the process is running. An empty string is returned if the domain is not set.

Errors: None.

See also: `GetHostName` ([220](#)), `seemGetdomainname2`

Listing: linuxex/ex39.pp

Program Example39;

{ Program to demonstrate the GetDomainName function. }

Uses linux;

begin

WriteLn ('Domain name of this machine is : ', GetDomainName);
end.

GetEGid

Declaration: `Function GetEGid : Longint;`

Description: Get the effective group ID of the currently running process.

Errors: None.

See also: `GetGid` ([220](#)), `getegid` (2)

Listing: linuxex/ex18.pp

Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses linux;

begin

writeLn ('Group Id = ', getgid, ' Effective group Id = ', getegid);
end.

GetEUid

Declaration: `Function GetEUid : Longint;`

Description: Get the effective user ID of the currently running process.

Errors: None.

See also: `GetEUid` ([218](#)), `geteuid` (2)

Listing: linuxex/ex17.pp

Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses linux;

begin

writeLn ('User Id = ', getuid, ' Effective user Id = ', geteuid);
end.

GetEnv

Declaration: `Function GetEnv (P : String) : PChar;`

Description: Returns the value of the environment variable in P. If the variable is not defined, nil is returned. The value of the environment variable may be the empty string. A PChar is returned to accomodate for strings longer than 255 bytes, TERMCAP and LS_COLORS, for instance.

Errors: None.

See also: `sh(1)`, `csh(1)`

Listing: `linuxex/ex41.pp`

Program `Example41;`

{ Program to demonstrate the GetEnv function. }

Uses `linux;`

begin

`WriteLn ('Path is : ', Getenv('PATH'));`
end.

GetEpochTime

Declaration: `Function GetEpochTime : longint;`

Description: returns the number of seconds since 00:00:00 gmt, january 1, 1970. it is adjusted to the local time zone, but not to DST.

Errors: no errors

See also: `EpochToLocal` ([201](#)), `GetTime` ([222](#)), `time` (2)

Listing: `linuxex/ex1.pp`

Program `Example1;`

{ Program to demonstrate the GetEpochTime function. }

Uses `linux;`

begin

`Write ('Secs past the start of the Epoch (00:00 1/1/1980) : ');`
`WriteLn (GetEpochTime);`
end.

GetFS

Declaration: `Function GetFS (Var F : Any File Type) : Longint;`

Description: `GetFS` returns the file selector that the kernel provided for your file. In principle you don't need this file selector. Only for some calls it is needed, such as the `Select` ([240](#)) call or so.

Errors: In case the file was not opened, then -1 is returned.

See also: `Select` ([240](#))

Listing: linuxex/ex34.pp

Program Example33;

```
{ Program to demonstrate the SelectText function. }

Uses linux;

Var tv : TimeVal;

begin
  Writeln ( 'Press the <ENTER> to continue the program.' );
  { Wait until File descriptor 0 (=Input) changes }
  SelectText ( Input, nil );
  { Get rid of <ENTER> in buffer }
  readln;
  Writeln ( 'Press <ENTER> key in less than 2 seconds...' );
  tv.sec:=2;
  tv.usec:=0;
  if SelectText ( Input, @tv)>0 then
    Writeln ( 'Thank you !' )
  else
    Writeln ( 'Too late !' );
end.
```

GetGid

Declaration: `Function GetGid : Longint;`

Description: Get the real group ID of the currently running process.

Errors: None.

See also: `GetEGid` (218), `getgid` (2)

Listing: linuxex/ex18.pp

Program Example18;

```
{ Program to demonstrate the GetGid and GetEGid functions. }

Uses linux;

begin
  writeln ( 'Group Id = ', getgid, ' Effective group Id = ', getegid );
end.
```

GetHostName

Declaration: `Function GetHostName : String;`

Description: Get the hostname of the machine on which the process is running. An empty string is returned if hostname is not set.

Errors: None.

See also: `GetDomainName` (217), `seemGethostname2`

Listing: linuxex/ex40.pp

Program Example40;

```
{ Program to demonstrate the GetHostName function. }
```

Uses linux;

begin

```
  Writeln ( 'Name of this machine is : ',GetHostName);  
end.
```

GetLocalTimezone

Declaration: `procedure GetLocalTimezone(timer:longint;var leap_correct,leap_hit:longint);
procedure GetLocalTimezone(timer:longint);`

Description: GetLocalTimezone returns the local timezone information. It also initializes the TZSeconds variable, which is used to correct the epoch time to local time.

There should never be any need to call this function directly. It is called by the initialization routines of the Linux unit.

See also: GetTimezoneFile ([223](#)), ReadTimezoneFile ([239](#))

GetPid

Declaration: `Function GetPid : Longint;`

Description: Get the Process ID of the currently running process.

Errors: None.

See also: GetPPid ([221](#)), getpid (2)

Listing: linuxex/ex16.pp

Program Example16;

```
{ Program to demonstrate the GetPid, GetPPid function. }
```

Uses linux;

begin

```
  Writeln ( 'Process Id = ',getpid, ' Parent process Id = ',getppid);  
end.
```

GetPPid

Declaration: `Function GetPPid : Longint;`

Description: Get the Process ID of the parent process.

Errors: None.

See also: GetPid ([221](#)), getppid (2)

Listing: linuxex/ex16.pp

Program Example16;

```
{ Program to demonstrate the GetPid, GetPPid function. }
```

Uses linux;

begin

```
  WriteLn ( 'Process Id = ',getpid,' Parent process Id = ',getppid);  
end.
```

GetPriority

Declaration: Function GetPriority (Which,Who : Integer) : Integer;

Description: GetPriority returns the priority with which a process is running. Which process(es) is determined by the Which and Who variables. Which can be one of the pre-defined Prio_Process, Prio_PGrp, Prio_User, in which case Who is the process ID, Process group ID or User ID, respectively.

Errors: Error checking must be done on LinuxError, since a priority can be negative.

sys_esrchNo process found using which and who.

sys_einvalWhich was not one of Prio_Process, Prio_Grp or Prio_User.

See also: SetPriority ([241](#)), Nice ([233](#)), Getpriority ([2](#))

For an example, see Nice ([233](#)).

GetTime

Declaration: procedure GetTime(var hour,min,sec,msec,usec:word); procedure GetTime(var hour,min,sec,sec100:word); procedure GetTime(var hour,min,sec:word);

Description: Returns the current time of the day, adjusted to local time. Upon return, the parameters are filled with

hourHours since 00:00 today.

minminutes in current hour.

secseconds in current minute.

sec100hundreds of seconds in current second.

msecmilliseconds in current second.

usecmicroseconds in current second.

Errors: None

See also: GetEpochTime ([219](#)), GetDate ([217](#)), GetDateTime ([217](#)), EpochToLocal ([201](#))

Listing: linuxex/ex5.pp

```
Program Example5;

{ Program to demonstrate the GetTime function. }

Uses linux;

Var Hour, Minute, Second : Word;

begin
    GetTime (Hour, Minute, Second);
    Writeln ( 'Time : ', Hour:2, ':', Minute:2, ':', Second:2);
end.
```

GetTimeOfDay

Declaration: Procedure GetTimeOfDay(var tv:timeval);

Description: GetTimeOfDay returns the number of seconds since 00:00, January 1 1970, GMT in a timeval record. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call. To get the local time, [GetTime \(222\)](#).

Errors: None.

See also: [GetTime \(222\)](#), [GetTimeOfDay \(223\)](#)

GetTimeOfDay

Declaration: Function GetTimeOfDay:longint;

Description: GetTimeOfDay returns the number of seconds since 00:00, January 1 1970, GMT. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call. To get the local time, [GetTime \(222\)](#).

Errors: None.

See also: [GetTimeOfDay \(223\)](#), [GetTime \(222\)](#)

GetTimezoneFile

Declaration: function GetTimezoneFile:string;

Description: GetTimezoneFile returns the location of the current timezone file. The location of file is determined as follows:

- 1.If /etc/timezone exists, it is read, and the contents of this file is returned. This should work on Debian systems.
- 2.If /usr/lib/zoneinfo/localtime exists, then it is returned. (this file is a symlink to the timezone file on SuSE systems)
- 3.If /etc/localtime exists, then it is returned. (this file is a symlink to the timezone file on RedHat systems)

Errors: If no file was found, an empty string is returned.

See also: [ReadTimezoneFile \(239\)](#)

GetUid

Declaration: `Function GetUid : Longint;`

Description: Get the real user ID of the currently running process.

Errors: None.

See also: [GetEUid \(218\)](#), [getuid \(2\)](#)

Listing: `linuxex/ex17.pp`

Program `Example17;`

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses `linux;`

```
begin
  writeln ('User Id = ',getuid,' Effective user Id = ',geteuid);
end.
```

Glob

Declaration: `Function Glob (Const Path : Pathstr) : PGlob;`

Description: Glob returns a pointer to a glob structure which contains all filenames which exist and match the pattern in Path. The pattern can contain wildcard characters, which have their usual meaning.

Errors: Returns nil on error, and `LinuxError` is set.

sys_enomemNo memory on heap for glob structure.

othersAs returned by the `opendir` call, and `sys_readdir`.

See also: [GlobFree \(225\)](#), [Glob \(3\)](#)

Listing: `linuxex/ex49.pp`

Program `Example49;`

{ Program to demonstrate the Glob and GlobFree functions. }

Uses `linux;`

Var `G1,G2 : PGlob;`

```
begin
  G1:=Glob ( '*' );
  if LinuxError=0 then
    begin
      G2:=G1;
      writeln ('Files in this directory : ');
      While g2<>Nil do
        begin
          writeln (g2^.name);
          g2:=g2^.next;
        end;
      GlobFree (g1);
    end;
end.
```

GlobFree

Declaration: `Procedure GlobFree (Var P : Pglob);`

Description: Releases the memory, occupied by a pglob structure. P is set to nil.

Errors: None

See also: Glob ([224](#))

For an example, see Glob ([224](#)).

IOCtl

Declaration: `Procedure IOCtl (Handle, Ndx: Longint; Data: Pointer);`

Description: This is a general interface to the Unix/ LINUX ioctl call. It performs various operations on the filedescriptor Handle. Ndx describes the operation to perform. Data points to data needed for the Ndx function. The structure of this data is function-dependent, so we don't elaborate on this here. For more information on this, see various manual pages under linux.

Errors: Errors are reported in LinuxError. They are very dependent on the used function, that's why we don't list them here

See also: ioctl (2)

Listing: linuxex/ex54.pp

Program Example54;

uses Linux;

{ Program to demonstrate the IOCtl function. }

var

tios : Termios;

begin

IOCtl(1, TCGETS, @tios);

WriteLn('Input Flags : \$', hexstr(tios.c_iflag, 8));

WriteLn('Output Flags : \$', hexstr(tios.c_oflag, 8));

WriteLn('Line Flags : \$', hexstr(tios.c_lflag, 8));

WriteLn('Control Flags: \$', hexstr(tios.c_cflag, 8));

end.

IOperm

Declaration: `Function IOperm (From, Num : Cardinal; Value : Longint) : boolean;`

Description: IOperm sets permissions on Num ports starting with port From to Value. The function returns True if the call was successful, False otherwise. *Remark:*

- This works ONLY as root.
- Only the first 0x03ff ports can be set.
- When doing a Fork ([216](#)), the permissions are reset. When doing a Execve ([204](#)) they are kept.

Errors: Errors are returned in LinuxError

See also: ioperm (2)

IsATTY

Declaration: `Function IsATTY (var f) : Boolean;`

Description: Check if the filehandle described by `f` is a terminal. `f` can be of type

1. `longint` for file handles;
2. Text for text variables such as input etc.

Returns `True` if `f` is a terminal, `False` otherwise.

Errors: No errors are reported

See also: [IOCtl \(225\)](#), [TTYName \(251\)](#)

S_ISBLK

Declaration: `Function S_ISBLK (m:integer) : boolean;`

Description: `S_ISBLK` checks the file mode `m` to see whether the file is a block device file. If so it returns `True`.

Errors: [FStat \(214\)](#), [S_ISLNK \(227\)](#), [S_ISREG \(227\)](#), [S_ISDIR \(226\)](#), [S_ISCHR \(226\)](#), [S_ISFIFO \(226\)](#), [S_ISSOCK \(227\)](#)

See also: [ISLNK](#).

S_ISCHR

Declaration: `Function S_ISCHR (m:integer) : boolean;`

Description: `S_ISCHR` checks the file mode `m` to see whether the file is a character device file. If so it returns `True`.

Errors: [FStat \(214\)](#), [S_ISLNK \(227\)](#), [S_ISREG \(227\)](#), [S_ISDIR \(226\)](#), [S_ISBLK \(226\)](#), [S_ISFIFO \(226\)](#), [S_ISSOCK \(227\)](#)

See also: [ISLNK](#).

S_ISDIR

Declaration: `Function S_ISDIR (m:integer) : boolean;`

Description: `S_ISDIR` checks the file mode `m` to see whether the file is a directory. If so it returns `True`

Errors: [FStat \(214\)](#), [S_ISLNK \(227\)](#), [S_ISREG \(227\)](#), [S_ISCHR \(226\)](#), [S_ISBLK \(226\)](#), [S_ISFIFO \(226\)](#), [S_ISSOCK \(227\)](#)

See also: [ISLNK](#).

S_ISFIFO

Declaration: `Function S_ISFIFO (m:integer) : boolean;`

Description: `S_ISFIFO` checks the file mode `m` to see whether the file is a fifo (a named pipe). If so it returns `True`.

Errors: [FStat \(214\)](#), [S_ISLNK \(227\)](#), [S_ISREG \(227\)](#), [S_ISDIR \(226\)](#), [S_ISCHR \(226\)](#), [S_ISBLK \(226\)](#), [S_ISSOCK \(227\)](#)

See also: [ISLNK](#).

S_ISLNK

Declaration: `Function S_ISLNK (m:integer) : boolean;`

Description: `S_ISLNK` checks the file mode `m` to see whether the file is a symbolic link. If so it returns `True`

Errors: `FStat` (214), `S_ISREG` (227), `S_ISDIR` (226), `S_ISCHR` (226), `S_ISBLK` (226), `S_ISFIFO` (226), `S_ISSOCK` (227)

See also:

Listing: `linuxex/ex53.pp`

Program `Example53;`

{ Program to demonstrate the S_ISLNK function. }

Uses `linux;`

Var `Info : Stat;`

begin

if `LStat (paramstr(1),info)` **then**

begin

if `S_ISLNK(info.mode)` **then**

Writeln ('File is a link');

if `S_ISREG(info.mode)` **then**

Writeln ('File is a regular file');

if `S_ISDIR(info.mode)` **then**

Writeln ('File is a directory');

if `S_ISCHR(info.mode)` **then**

Writeln ('File is a character device file');

if `S_ISBLK(info.mode)` **then**

Writeln ('File is a block device file');

if `S_ISFIFO(info.mode)` **then**

Writeln ('File is a named pipe (FIFO)');

if `S_ISSOCK(info.mode)` **then**

Writeln ('File is a socket');

end;

end.

S_ISREG

Declaration: `Function S_ISREG (m:integer) : boolean;`

Description: `S_ISREG` checks the file mode `m` to see whether the file is a regular file. If so it returns `True`

Errors: `FStat` (214), `S_ISLNK` (227), `S_ISDIR` (226), `S_ISCHR` (226), `S_ISBLK` (226), `S_ISFIFO` (226), `S_ISSOCK` (227)

See also: `ISLNK`.

S_ISSOCK

Declaration: `Function S_ISSOCK (m:integer) : boolean;`

Description: `S_ISSOCK` checks the file mode `m` to see whether the file is a socket. If so it returns `True`.

Errors: FStat (214), S_ISLNK (227), S_ISREG (227), S_ISDIR (226), S_ISCHR (226), S_ISBLK (226), S_ISFIFO (226)

See also: ISLNK.

Kill

Declaration: Function Kill (Pid : Longint; Sig : Integer) : Integer;

Description: Send a signal Sig to a process or process group. If Pid>0 then the signal is sent to Pid, if it equals -1, then the signal is sent to all processes except process 1. If Pid<-1 then the signal is sent to process group -Pid. The return value is zero, except in case three, where the return value is the number of processes to which the signal was sent.

Errors: LinuxError is used to report errors:

sys_einvalAn invalid signal is sent.

sys_esrchThe Pid or process group don't exist.

sys_epermThe effective userid of the current process doesn't math the one of process Pid.

See also: SigAction (242), Signal (245), Kill (2)

LStat

Declaration: Function LStat (Path : Pathstr; Var Info : stat) : Boolean;

Description: LStat gets information about the link specified in Path, and stores it in Info, which is of type stat. Contrary to FStat, it stores information about the link, not about the file the link points to. The function returns True if the call was succesfull, False if the call failed.

Errors: LinuxError is used to report errors.

sys_enoentPath does not exist.

See also: FStat (214), FSStat (213), stat (2)

Listing: linuxex/ex29.pp

```
program example29;

{ Program to demonstrate the LStat function. }

uses linux;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f, 'test.fil');
  rewrite (f);
  for i:=1 to 10 do writeln (f, 'Testline # ', i);
  close (f);
  { Do the call on made file. }
  if not fstat ('test.fil', info) then
    begin
```

```
        writeln('Fstat failed. Errno : ',linuxerror);
        halt (1);
    end;
writeln;
writeln ('Result of fstat on file ''test.fil''.');
writeln ('Inode   : ',info.ino);
writeln ('Mode    : ',info.mode);
writeln ('nlink   : ',info.nlink);
writeln ('uid     : ',info.uid);
writeln ('gid     : ',info.gid);
writeln ('rdev    : ',info.rdev);
writeln ('Size    : ',info.size);
writeln ('Blksize  : ',info.blksize);
writeln ('Blocks   : ',info.blocks);
writeln ('atime    : ',info.atime);
writeln ('mtime    : ',info.mtime);
writeln ('ctime    : ',info.ctime);

If not SymLink ('test.fil','test.lnk') then
    writeln ('Link failed ! Errno : ',linuxerror);

if not lstat ('test.lnk',info) then
    begin
        writeln('LStat failed. Errno : ',linuxerror);
        halt (1);
    end;
writeln;
writeln ('Result of fstat on file ''test.lnk''.');
writeln ('Inode   : ',info.ino);
writeln ('Mode    : ',info.mode);
writeln ('nlink   : ',info.nlink);
writeln ('uid     : ',info.uid);
writeln ('gid     : ',info.gid);
writeln ('rdev    : ',info.rdev);
writeln ('Size    : ',info.size);
writeln ('Blksize  : ',info.blksize);
writeln ('Blocks   : ',info.blocks);
writeln ('atime    : ',info.atime);
writeln ('mtime    : ',info.mtime);
writeln ('ctime    : ',info.ctime);
{ Remove file and link }
erase (f);
unlink ('test.lnk');
end.
```

Link

Declaration: Function Link (OldPath,NewPath : pathstr) : Boolean;

Description: Link makes NewPath point to the same file als OldPath. The two files then have the same inode number. This is known as a 'hard' link. The function returns True if the call was succesfull, False if the call failed.

Errors: Errors are returned in LinuxError.

sys_exdevOldPath and NewPath are not on the same filesystem.

sys_epermThe filesystem containing oldpath and newpath doesn't support linking files.

sys_eaccess Write access for the directory containing Newpath is disallowed, or one of the directories in OldPath or NewPath has no search (=execute) permission.

sys_enoent A directory entry in OldPath or NewPath does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in OldPath or NewPath is not a directory.

sys_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only filesystem.

sys_eexist NewPath already exists.

sys_mlink OldPath has reached maximal link count.

sys_eloop OldPath or NewPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospc The device containing NewPath has no room for another entry.

sys_eperm OldPath points to . or .. of a directory.

See also: SymLink ([246](#)), UnLink ([252](#)), Link ([2](#))

Listing: linux/ex21.pp

Program Example21;

{ Program to demonstrate the Link and UnLink functions. }

Uses linux;

Var F : Text;

 S : **String**;

begin

 Assign (F, 'test.txt');

Rewrite (F);

WriteLn (F, 'This is written to test.txt');

 Close(f);

{ new.txt and test.txt are now the same file }

if not Link ('test.txt', 'new.txt') **then**

writeln ('Error when linking !');

{ Removing test.txt still leaves new.txt }

If not Unlink ('test.txt') **then**

WriteLn ('Error when unlinking !');

 Assign (f, 'new.txt');

Reset (F);

While not EOF(f) **do**

begin

ReadLn(F,S);

WriteLn ('> ',s);

end;

 Close (f);

{ Remove new.txt also }

If not Unlink ('new.txt') **then**

WriteLn ('Error when unlinking !');

end.

LocalToEpoch

Declaration: Function LocalToEpoch (Year,Month,Day,Hour,Minute,Second : Word) : longint;

Description: Converts the Local time to epoch time (=Number of seconds since 00:00:00 , January 1, 1970).

Errors: None

See also: [GetEpochTime \(219\)](#), [EpochToLocal \(201\)](#), [GetTime \(222\)](#), [GetDate \(217\)](#)

Listing: linuxex/ex4.pp

Program Example4;

{ Program to demonstrate the LocalToEpoch function . }

Uses linux;

Var year, month, day, hour, minute, second : Word;

begin

Write ('Year : '); **readln** (Year);

Write ('Month : '); **readln** (Month);

Write ('Day : '); **readln** (Day);

Write ('Hour : '); **readln** (Hour);

Write ('Minute : '); **readln** (Minute);

Write ('Seonds : '); **readln** (Second);

Write ('This is : ');

Write (LocalToEpoch (year, month, day, hour, minute, second));

Writeln (' seconds past 00:00 1/1/1980');

end.

MkFifo

Declaration: `Function MkFifo (PathName: String; Mode : Longint) : Boolean;`

Description: `MkFifo` creates a named pipe in the filesystem, with name `PathName` and mode `Mode`.

Errors: `LinuxError` is used to report errors:

sys_enfileToo many file descriptors for this process.

sys_enfileThe system file table is full.

See also: [POpen \(236\)](#), [MkFifo \(231\)](#), `mkfifo` (4)

MMap

Declaration: `Function MMap(const m:tmmmapargs):longint;`

Description: `MMap` maps or unmaps files or devices into memory. The different fields of the argument `m` determine what and how the `mmap` maps this:

addressAddress where to `mmap` the device. This address is a hint, and may not be followed.

sizeSize (in bytes) of area to be mapped.

protProtection of mapped memory. This is a OR-ed combination of the following constants:

PROT_EXECThe memory can be executed.

PROT_READThe memory can be read.

PROT_WRITEThe memory can be written.

PROT_NONEThe memory can not be accessed.

flags Contains some options for the `mmap` call. It is an OR-ed combination of the following constants:

MAP_FIXED Do not map at another address than the given address. If the address cannot be used, `MMap` will fail.

MAP_SHARED Share this map with other processes that map this object.

MAP_PRIVATE Create a private map with copy-on-write semantics.

MAP_ANONYMOUS `fd` does not have to be a file descriptor.

One of the options `MAP_SHARED` and `MAP_PRIVATE` must be present, but not both at the same time.

fd File descriptor from which to map.

offset Offset to be used in file descriptor `fd`.

The function returns a pointer to the mapped memory, or a -1 in case of an error.

Errors: On error, -1 is returned and `LinuxError` is set to the error code:

Sys_EBADF `fd` is not a valid file descriptor and `MAP_ANONYMOUS` was not specified.

Sys_EACCES `MAP_PRIVATE` was specified, but `fd` is not open for reading. Or `MAP_SHARED` was asked and `PROT_WRITE` is set, `fd` is not open for writing

Sys_EINVAL One of the record fields `Start`, `length` or `offset` is invalid.

Sys_ETXTBUSY `MAP_DENYWRITE` was set but the object specified by `fd` is open for writing.

Sys_EAGAIN `fd` is locked, or too much memory is locked.

Sys_ENOMEM Not enough memory for this operation.

See also: `MUnMap` (233), `mmap` (2)

Listing: `linuxex/ex66.pp`

Program `Example66`;

{ Program to demonstrate the MMap function. }

Uses `linux`;

Var `S` : **String**;

`fd, Len` : **Longint**;

`args` : **tmapargs**;

`P` : **PChar**;

begin

`S := 'This is a string' #0;`

`Len := Length(S);`

`fd := fdOpen('testfile.txt', Open_wrOnly or open_creat);`

If `fd = -1` **then**

Halt(1);

If `fdWrite(fd, S[1], Len) = -1` **then**

Halt(2);

`fdClose(fd);`

`fdOpen('testfile.txt', Open_rdOnly);`

if `fd = -1` **then**

Halt(3);

`args.address := 0;`

`args.offset := 0;`

`args.size := Len + 1;`

`args.fd := fd;`

`args.flags := MAP_PRIVATE;`

```
args.prot:=PROT_READ or PROT_WRITE;
P:=Pchar(mmap(args));
If longint(P)=-1 then
  Halt(4);
Writeln('Read in memory  :',P);
fdclose(fd);
if Not MUnMap(P,Len) Then
  Halt(LinuxError);
end.
```

MUnMap

Declaration: `function MUnMap (P : Pointer; Size : Longint) : Boolean;`

Description: MUnMap unmaps the memory block of size Size, pointed to by P, which was previously allocated with MMap (231).

The function returns True if successful, False otherwise.

Errors: In case of error the function returns False and LinuxError is set to an error value. See MMap (231) for possible error values.

See also: MMap (231), munmap (2)

For an example, see MMap (231).

Nice

Declaration: `Procedure Nice (N : Integer);`

Description: Nice adds -N to the priority of the running process. The lower the priority numerically, the less the process is favored. Only the superuser can specify a negative N, i.e. increase the rate at which the process is run.

Errors: Errors are returned in LinuxError

sys_epermA non-superuser tried to specify a negative N, i.e. do a priority increase.

See also: GetPriority (222), SetPriority (241), Nice (2)

Listing: linuxex/ex15.pp

Program Example15;

{ Program to demonstrate the Nice and Get/SetPriority functions. }

Uses linux;

begin

```
  writeln ('Setting priority to 5');
  setpriority (prio_process,getpid,5);
  writeln ('New priority = ',getpriority (prio_process,getpid));
  writeln ('Doing nice 10');
  nice (10);
  writeln ('New Priority = ',getpriority (prio_process,getpid));
end.
```

Octal

Declaration: `Function Octal(l:longint):longint;`

Description: `Octal` will convert a number specified as an octal number to its decimal value.

This is useful for the `Chmod` ([195](#)) call, where permissions are specified as octal numbers.

Errors: No checking is performed whether the given number is a correct Octal number. e.g. specifying 998 is possible; the result will be wrong in that case.

See also: `Chmod` ([195](#)).

Listing: `linuxex/ex68.pp`

Program `Example68;`

{ Program to demonstrate the Octal function. }

Uses `linux;`

begin

`Writeln('Mode 777 : ', Octal(777));`

`Writeln('Mode 644 : ', Octal(644));`

`Writeln('Mode 755 : ', Octal(755));`

end.

OpenDir

Declaration: `Function OpenDir (f:pchar) : pdir; Function OpenDir (f:string) : pdir;`

Description: `OpenDir` opens the directory `f`, and returns a `pdir` pointer to a `Dir` record, which can be used to read the directory structure. If the directory cannot be opened, `nil` is returned.

Errors: Errors are returned in `LinuxError`.

See also: `CloseDir` ([198](#)), `ReadDir` ([236](#)), `SeekDir` ([239](#)), `TellDir` ([251](#)), `opendir` ([3](#))

Listing: `linuxex/ex35.pp`

Program `Example35;`

*{ Program to demonstrate the
 OpenDir, ReadDir, SeekDir and TellDir functions. }*

Uses `linux;`

Var `TheDir : PDir;`
`ADirent : PDirent;`
`Entry : Longint;`

begin

`TheDir:=OpenDir('./.');`

Repeat

`Entry:=TellDir(TheDir);`

`ADirent:=ReadDir (TheDir);`

If `ADirent<>Nil` **then**

With `ADirent^` **do**

begin

```
        Writeln ( ' Entry No : ', Entry );
        Writeln ( ' Inode    : ', ino );
        Writeln ( ' Offset   : ', off );
        Writeln ( ' Reclen   : ', reclen );
        Writeln ( ' Name     : ', pchar(@name[0]));
    end;
Until ADirent=Nil;
Repeat
    Write ( ' Entry No. you would like to see again (-1 to stop): ' );
    ReadLn ( Entry );
    If Entry<>-1 then
    begin
        SeekDir ( TheDir, Entry );
        ADirent:=ReadDir ( TheDir );
        If ADirent<>Nil then
            With ADirent^ do
            begin
                Writeln ( ' Entry No : ', Entry );
                Writeln ( ' Inode    : ', ino );
                Writeln ( ' Offset   : ', off );
                Writeln ( ' Reclen   : ', reclen );
                Writeln ( ' Name     : ', pchar(@name[0]));
            end;
        end;
    Until Entry=-1;
    CloseDir ( TheDir );
end.
```

pause

Declaration: Procedure Pause;

Description: Pause puts the process to sleep and waits until the application receives a signal. If a signal handler is installed for the received sigal, the handler will be called and after that pause will return control to the process.

Errors: None.

For an example, see Alarm ([190](#)).

PClose

Declaration: Function PClose (Var F : FileType) : longint;

Description: PClose closes a file opened with POpen. It waits for the command to complete, and then returns the exit status of the command.

Errors: LinuxError is used to report errors. If it is different from zero, the exit status is not valid.

See also: POpen ([236](#))

For an example, see POpen ([236](#))

POpen

Declaration: Procedure POpen (Var F : FileType; Cmd : pathstr; rw : char);

Description: Popen runs the command specified in Cmd, and redirects the standard in or output of the command to the other end of the pipe F. The parameter rw indicates the direction of the pipe. If it is set to 'W', then F can be used to write data, which will then be read by the command from stdin. If it is set to 'R', then the standard output of the command can be read from F. F should be reset or rewritten prior to using it. F can be of type Text or File. A file opened with POpen can be closed with Close, but also with PClose (235). The result is the same, but PClose returns the exit status of the command Cmd.

Errors: Errors are reported in LinuxError and are essentially those of the Execve, Dup and AssignPipe commands.

See also: AssignPipe (190), popen (3), PClose (235)

Listing: linuxex/ex37.pp

Program Example37;

```
{ Program to demonstrate the Popen function. }

uses linux;

var f : text;
    i : longint;

begin
  writeln ('Creating a shell script to which echoes its arguments');
  writeln ('and input back to stdout');
  assign (f, 'test21a');
  rewrite (f);
  writeln (f, '#!/bin/sh');
  writeln (f, 'echo this is the child speaking.... ');
  writeln (f, 'echo got arguments \*"${*}"\*');
  writeln (f, 'cat');
  writeln (f, 'exit 2');
  writeln (f);
  close (f);
  chmod ('test21a', octal (755));
  popen (f, './test21a arg1 arg2', 'W');
  if linuxerror<>0 then
    writeln ('error from POpen : Linuxerror : ', Linuxerror);
  for i:=1 to 10 do
    writeln (f, 'This is written to the pipe, and should appear on stdout. ');
  Flush(f);
  Writeln ('The script exited with status : ', PClose (f));
  writeln;
  writeln ('Press <return> to remove shell script. ');
  readln;
  assign (f, 'test21a');
  erase (f)
end.
```

ReadDir

Declaration: Function ReadDir (p:pdirent) : pdirent;

Description: `ReadDir` reads the next entry in the directory pointed to by `p`. It returns a `pdirent` pointer to a structure describing the entry. If the next entry can't be read, `Nil` is returned.

Errors: Errors are returned in `LinuxError`.

See also: `CloseDir` (198), `OpenDir` (234), `SeekDir` (239), `TellDir` (251), `readdir` (3)

For an example, see `OpenDir` (234).

ReadLink

Declaration: `Function ReadLink(name,linkname:pchar;maxlen:longint):longint; Function ReadLink(name:pathstr):pathstr;`

Description: `ReadLink` returns the file the symbolic link name is pointing to. The first form of this function accepts a buffer `linkname` of length `maxlen` where the filename will be stored. It returns the actual number of characters stored in the buffer.

The second form of the function returns simply the name of the file.

Errors: On error, the first form of the function returns -1; the second one returns an empty string. `LinuxError` is set to report errors:

SYS_ENOTDIRA part of the path in `Name` is not a directory.

SYS_EINVAL`maxlen` is not positive, or the file is not a symbolic link.

SYS_ENAMETOOLONGA pathname, or a component of a pathname, was too long.

SYS_ENOENTthe link name does not exist.

SYS_EACCESNo permission to search a directory in the path

SYS_ELOOPToo many symbolic links were encountered in translating the pathname.

SYS_EIOAn I/O error occurred while reading from the file system.

SYS_EFAULTThe buffer is not part of the process's memory space.

SYS_ENOMEMNot enough kernel memory was available.

See also: `SymLink` (246)

Listing: `linuxex/ex62.pp`

Program `Example62;`

{ Program to demonstrate the ReadLink function. }

Uses `linux;`

Var `F : Text;`
 `S : String;`

begin
 `Assign (F, 'test.txt');`
 `Rewrite (F);`
 `Writeln (F, 'This is written to test.txt');`
 `Close(f);`
 { new.txt and test.txt are now the same file }
 if not `SymLink ('test.txt', 'new.txt')` **then**
 `writeln ('Error when symlinking !');`
 `S:=ReadLink('new.txt');`

```
  If S='' then
    Writeln ('Error reading link !')
  Else
    Writeln ('Link points to : ',S);
  { Now remove links }
  If not Unlink ('new.txt') then
    Writeln ('Error when unlinking !');
  If not Unlink ('test.txt') then
    Writeln ('Error when unlinking !');
end.
```

ReadPort

Declaration: Procedure ReadPort (Port : Longint; Var Value : Byte); Procedure ReadPort (Port : Longint; Var Value : Word); Procedure ReadPort (Port : Longint; Var Value : Longint);

Description: ReadPort reads one Byte, Word or Longint from port Port into Value.

Note that you need permission to read a port. This permission can be set by the root user with the [IOperm \(225\)](#) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: [IOperm \(225\)](#), [ReadPortB \(238\)](#), [ReadPortW \(239\)](#), [ReadPortL \(238\)](#), [WritePort \(254\)](#), [WritePortB \(254\)](#), [WritePortL \(254\)](#), [WritePortW \(255\)](#)

ReadPortB

Declaration: Procedure ReadPortB (Port : Longint; Var Buf; Count: longint); Function ReadPortB (Port : Longint): Byte;

Description: The procedural form of ReadPortB reads Count bytes from port Port and stores them in Buf. There must be enough memory allocated at Buf to store Count bytes.

The functional form of ReadPortB reads 1 byte from port B and returns the byte that was read.

Note that you need permission to read a port. This permission can be set by the root user with the [IOperm \(225\)](#) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: [IOperm \(225\)](#), [ReadPort \(238\)](#), [ReadPortW \(239\)](#), [ReadPortL \(238\)](#), [WritePort \(254\)](#), [WritePortB \(254\)](#), [WritePortL \(254\)](#), [WritePortW \(255\)](#)

ReadPortL

Declaration: function ReadPortL (Port : Longint): LongInt; Procedure ReadPortL (Port : Longint; Var Buf; Count: longint);

Description: The procedural form of ReadPortL reads Count longints from port Port and stores them in Buf. There must be enough memory allocated at Buf to store Count Longints.

The functional form of ReadPortB reads 1 longint from port B and returns the longint that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `IOperm` (225) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `IOperm` (225), `ReadPort` (238), `ReadPortW` (239), `ReadPortB` (238), `WritePort` (254), `WritePortB` (254), `WritePortL` (254), `WritePortW` (255)

ReadPortW

Declaration: `Procedure ReadPortW (Port : Longint; Var Buf; Count: longint); function ReadPortW (Port : Longint): Word;`

Description: The procedural form of `ReadPortB` reads `Count` words from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` words.

The functional form of `ReadPortB` reads 1 word from port `B` and returns the word that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `IOperm` (225) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `IOperm` (225), `ReadPort` (238), `ReadPortB` (238), `ReadPortL` (238), `WritePort` (254), `WritePortB` (254), `WritePortL` (254), `WritePortW` (255)

ReadTimezoneFile

Declaration: `procedure ReadTimezoneFile(fn:string);`

Description: `ReadTimezoneFile` reads the timezone file `fn` and initializes the local time routines based on the information found there.

There should be no need to call this function. The initialization routines of the `linux` unit call this routine at unit startup.

Errors: None.

See also: `GetTimezoneFile` (223), `GetLocalTimezone` (221)

SeekDir

Declaration: `Procedure SeekDir (p:pdir;off:longint);`

Description: `SeekDir` sets the directory pointer to the `off`-th entry in the directory structure pointed to by `p`.

Errors: Errors are returned in `LinuxError`.

See also: `CloseDir` (198), `ReadDir` (236), `OpenDir` (234), `TellDir` (251), `seekdir` (3)

For an example, see `OpenDir` (234).

Select

Declaration: Function Select (N : Longint;
var readfds, writefds, exceptfds : PFDset; Var Timeout) : Longint;

Description: Select checks one of the file descriptors in the FDsets to see if its status changed. readfds, writefds and exceptfds are pointers to arrays of 256 bits. If you want a file descriptor to be checked, you set the corresponding element in the array to 1. The other elements in the array must be set to zero. Three arrays are passed : The entries in readfds are checked to see if characters become available for reading. The entries in writefds are checked to see if it is OK to write to them, while entries in exceptfds are checked to see if an exception occurred on them. You can use the functions FD_ZERO (206), FD_Clr (206), FD_Set (207), FD_IsSet (206) to manipulate the individual elements of a set. The pointers can be nil. N is the largest index of a nonzero entry plus 1. (= the largest file-descriptor + 1). Timeout can be used to set a time limit. If Timeout can be two types :

1. Timeout is of type PTime and contains a zero time, the call returns immediately. If Timeout is Nil, the kernel will wait forever, or until a status changed.
2. Timeout is of type Longint. If it is -1, this has the same effect as a Timeout of type PTime which is Nil. Otherwise, Timeout contains a time in milliseconds.

When the Timeout is reached, or one of the file descriptors has changed, the Select call returns. On return, it will have modified the entries in the array which have actually changed, and it returns the number of entries that have been changed. If the timeout was reached, and no descriptor changed, zero is returned; The arrays of indexes are undefined after that. On error, -1 is returned.

Errors: On error, the function returns -1, and Errors are reported in LinuxError :

SYS_EBADF An invalid descriptor was specified in one of the sets.

SYS_EINTR A non blocked signal was caught.

SYS_EINVAL N is negative or too big.

SYS_ENOMEM Select was unable to allocate memory for its internal tables.

See also: SelectText (241), GetFS (219), FD_ZERO (206), FD_Clr (206), FD_Set (207), FD_IsSet (206)

Listing: linuxex/ex33.pp

Program Example33;

{ Program to demonstrate the Select function. }

Uses linux;

Var FDS : FDSet;

begin

```
FD_Zero (FDS);
FD_Set (0, FDS);
Writeln ('Press the <ENTER> to continue the program. ');
{ Wait until File descriptor 0 (=Input) changes }
Select (1, @FDS, nil, nil, nil);
{ Get rid of <ENTER> in buffer }
readln;
Writeln ('Press <ENTER> key in less than 2 seconds... ');
FD_Zero (FDS);
FD_Set (0, FDS);
if Select (1, @FDS, nil, nil, 2000) > 0 then
```

```
    WriteLn ( 'Thank you !' )  
    { FD_ISSET(0,FDS) would be true here. }  
  else  
    WriteLn ( 'Too late !' );  
end.
```

SelectText

Declaration: Function SelectText (var T : Text; TimeOut : PTime) : Longint;

Description: SelectText executes the [Select \(240\)](#) call on a file of type Text. You can specify a timeout in TimeOut. The SelectText call determines itself whether it should check for read or write, depending on how the file was opened : With Reset it is checked for reading, with Rewrite and Append it is checked for writing.

Errors: See [Select \(240\)](#). SYS_EBADF can also mean that the file wasn't opened.

See also: [Select \(240\)](#), [GetFS \(219\)](#)

SetPriority

Declaration: Function SetPriority (Which,Who,Prio : Integer) : Integer;

Description: SetPriority sets the priority with which a process is running. Which process(es) is determined by the Which and Who variables. Which can be one of the pre-defined Prio_Process, Prio_PGrp, Prio_User, in which case Who is the process ID, Process group ID or User ID, respectively. Prio is a value in the range -20 to 20.

Errors: Error checking must be done on LinuxError, since a priority can be negative.

sys_esrchNo process found using which and who.

sys_einvalWhich was not one of Prio_Process, Prio_Grp or Prio_User.

sys_epermA process was found, but neither its effective or real user ID match the effective user ID of the caller.

sys_eaccessA non-superuser tried to a priority increase.

See also: [GetPriority \(222\)](#), [Nice \(233\)](#), [Setpriority \(2\)](#)

For an example, see [Nice \(233\)](#).

Shell

Declaration: Function Shell (Command : String) : Longint;

Description: Shell invokes the bash shell (/bin/sh), and feeds it the command Command (using the -c option). The function then waits for the command to complete, and then returns the exit status of the command, or 127 if it could not complete the [Fork \(216\)](#) or [Execve \(204\)](#) calls.

Errors: Errors are reported in LinuxError.

See also: [POpen \(236\)](#), [Fork \(216\)](#), [Execve \(204\)](#), [system \(3\)](#)

Listing: linuxex/ex56.pp

```
program example56;

uses linux;

{ Program to demonstrate the Shell function }

Var S : Longint;

begin
  Writeln ( 'Output of ls -l *.pp' );
  S:=Shell ( 'ls -l *.pp' );
  Writeln ( 'Command exited with status : ',S);
end.
```

SigAction

Declaration: Procedure SigAction (Signalum : Integer; Var Act,OldAct : PSigActionRec);

Description: Changes the action to take upon receipt of a signal. Act and Oldact are pointers to a SigActionRec record. Signalum specifies the signal, and can be any signal except **SIGKILL** or **SIGSTOP**. If Act is non-nil, then the new action for signal Signalum is taken from it. If OldAct is non-nil, the old action is stored there. Sa_Handler may be SIG_DFL for the default action or SIG_IGN to ignore the signal. Sa_Mask Specifies which signals should be ignored during the execution of the signal handler. Sa_Flags Specifies a series of flags which modify the behaviour of the signal handler. You can 'or' none or more of the following :

SA_NOCLDSTOPIf signalum is **SIGCHLD** do not receive notification when child processes stop.

SA_ONESHOT or **SA_RESETHAND**Restore the signal action to the default state once the signal handler has been called.

SA_RESTARTFor compatibility with BSD signals.

SA_NOMASK or **SA_NODEFER**Do not prevent the signal from being received from within its own signal handler.

Errors: LinuxError is used to report errors.

sys_einvalan invalid signal was specified, or it was **SIGKILL** or **SIGSTOP**.

sys_efaultAct , OldAct point outside this process address space

sys_eintrSystem call was interrupted.

See also: SigProcMask ([243](#)), SigPending ([243](#)), SigSuspend ([244](#)), Kill ([228](#)), Sigaction ([2](#))

Listing: linuxex/ex57.pp

```
Program example57;

{ Program to demonstrate the SigAction function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

uses Linux;
```

```

Var
    oa,na : PSigActionRec;

Procedure DoSig(sig : Longint);cdecl;

begin
    writeln('Receiving signal: ', sig);
end;

begin
    new(na);
    new(oa);
    na^.Handler.sh:=@DoSig;
    na^.Sa_Mask:=0;
    na^.Sa_Flags:=0;
    na^.Sa_Restorer:=Nil;
    SigAction(SigUsrc1,na,oa);
    if LinuxError<>0 then
        begin
            writeln('Error: ',linuxerror,'. ');
            halt(1);
        end;
    Writeln ('Send USR1 signal or press <ENTER> to exit');
    readln;
end.

```

SigPending

Declaration: `Function SigPending : SigSet;`

Description: Sigpending allows the examination of pending signals (which have been raised while blocked.) The signal mask of pending signals is returned.

Errors: None

See also: SigAction (242), SigProcMask (243), SigSuspend (244), Signal (245), Kill (228), Sigpending (2)

SigProcMask

Declaration: `Procedure SigProcMask (How : Integer; SSet,OldSSet : PSigSet);`

Description: Changes the list of currently blocked signals. The behaviour of the call depends on How :

SIG_BLOCKThe set of blocked signals is the union of the current set and the SSet argument.

SIG_UNBLOCKThe signals in SSet are removed from the set of currently blocked signals.

SIG_SETMASKThe list of blocked signals is set so SSet.

If OldSSet is non-nil, then the old set is stored in it.

Errors: LinuxError is used to report errors.

sys_efaultSSet or OldSSet point to an adress outside the range of the process.

sys_eintrSystem call was interrupted.

See also: SigAction (242), SigPending (243), SigSuspend (244), Kill (228), Sigprocmask (2)

SigRaise

Declaration: `Procedure SigRaise(Sig:integer);`

Description: `SigRaise` sends a `Sig` signal to the current process.

Errors: None.

See also: [Kill \(228\)](#), [GetPid \(221\)](#)

Listing: `linuxex/ex65.pp`

Program `example64`;

{ Program to demonstrate the SigRaise function. }

uses `Linux`;

Var

`oa, na : PSigActionRec;`

Procedure `DoSig(sig : Longint); cdecl;`

begin

`writeln('Receiving signal: ', sig);`

end;

begin

`new(na);`

`new(oa);`

`na^.handler.sh:=@DoSig;`

`na^.Sa_Mask:=0;`

`na^.Sa_Flags:=0;`

`na^.Sa_Restorer:=Nil;`

`SigAction(SigUsr1, na, oa);`

if `LinuxError<>0` **then**

begin

`writeln('Error: ', linuxerror, '.');`

`halt(1);`

end;

`WriteLn('Sending USR1 (' , sigusr1, ') signal to self.');`

`SigRaise(sigusr1);`

end.

SigSuspend

Declaration: `Procedure SigSuspend (Mask : SigSet);`

Description: `SigSuspend` temporarily replaces the signal mask for the process with the one given in `Mask`, and then suspends the process until a signal is received.

Errors: None

See also: [SigAction \(242\)](#), [SigProcMask \(243\)](#), [SigPending \(243\)](#), [Signal \(245\)](#), [Kill \(228\)](#), [SigSuspend \(2\)](#)

Signal

Declaration: `Function Signal (SigNum : Integer; Handler : SignalHandler) : SignalHandler;`

Description: `Signal` installs a new signal handler for signal `SigNum`. This call has the same functionality as the `SigAction` call. The return value for `Signal` is the old signal handler, or `nil` on error.

Errors: `LinuxError` is used to report errors :

SIG_ERRAn error occurred.

See also: `SigAction` (242), `Kill` (228), `Signal` (2)

Listing: `linuxex/ex58.pp`

Program `example58;`

```
{ Program to demonstrate the Signal function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

uses Linux;

Procedure DoSig(sig : Longint); cdecl;

begin
  writeln('Receiving signal: ', sig);
end;

begin
  SigNal(SigUsr1, @DoSig);
  if LinuxError <> 0 then
    begin
      writeln('Error: ', linuxerror, '.');
      halt(1);
    end;
  Writeln ('Send USR1 signal or press <ENTER> to exit');
  readln;
end.
```

StringToPPchar

Declaration: `Function StringToPPChar(Var S:String):ppchar;`

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of `pchars` that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various `Exec` calls.

Errors: None.

See also: [CreateShellArgV \(198\)](#), [Execve \(204\)](#), [Execv \(203\)](#)

Listing: linuxex/ex70.pp

Program Example70;

{ Program to demonstrate the StringToPPchar function. }

Uses linux;

Var S : **String**;
P : PPChar;
I : longint;

begin
// remark whitespace **at end**.
S:='This is a string with words. ';
P:=StringToPPChar(S);
I:=0;
While P[I]<>Nil **do**
 begin
 WriteLn('Word ',I,' : ',P[I]);
 Inc(I);
 end;
 FreeMem(P,I***SizeOf**(Pchar));
end.

SymLink

Declaration: Function SymLink (OldPath,NewPath : pathstr) : Boolean;

Description: SymLink makes Newpath point to the file in OldPath, which doesn't necessarily exist. The two files DO NOT have the same inode number. This is known as a 'soft' link. The permissions of the link are irrelevant, as they are not used when following the link. Ownership of the file is only checked in case of removal or renaming of the link. The function returns True if the call was succesfull, False if the call failed.

Errors: Errors are returned in LinuxError.

sys_epermThe filesystem containing oldpath and newpath doesn't support linking files.

sys_eaccessWrite access for the directory containing Newpath is disallowed, or one of the directories in OldPath or NewPath has no search (=execute) permission.

sys_enoentA directory entry in OldPath or NewPath does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdirA directory entry in OldPath or NewPath is nor a directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe files are on a read-only filesystem.

sys_eexistNewPath already exists.

sys_eloopOldPath or NewPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospcThe device containing NewPath has no room for anothe entry.

See also: [Link \(229\)](#), [UnLink \(252\)](#), [ReadLink \(237\)](#), [SymLink \(2\)](#)

Listing: linuxex/ex22.pp

Program Example22;

```
{ Program to demonstrate the SymLink and UnLink functions. }
```

Uses linux;

Var F : Text;
 S : **String**;

```
begin  
  Assign (F, 'test.txt');  
  Rewrite (F);  
  WriteIn (F, 'This is written to test.txt');  
  Close(f);  
  { new.txt and test.txt are now the same file }  
  if not SymLink ('test.txt', 'new.txt') then  
    writeln ('Error when symlinking !');  
  { Removing test.txt still leaves new.txt  
    Pointing now to a non-existent file ! }  
  If not Unlink ('test.txt') then  
    WriteIn ('Error when unlinking !');  
  Assign (f, 'new.txt');  
  { This should fail, since the symbolic link  
    points to a non-existent file ! }  
  {$i-}  
  Reset (F);  
  {$i+}  
  If IOResult=0 then  
    WriteIn ('This shouldn't happen');  
  { Now remove new.txt also }  
  If not Unlink ('new.txt') then  
    WriteIn ('Error when unlinking !');  
end.
```

SysInfo

Declaration: Function SysInfo(var Info:TSysinfo):Boolean;

Description: SysInfo returns system information in Info. Returned information in Info includes:

- uptime**Number of seconds since boot.
- loads**1, 5 and 15 minute load averages.
- totalram**total amount of main memory.
- freeram**amount of free memory.
- sharedram**amount of shared memory
- bufferram**amount of memory used by buffers.
- totalswap**total amount of swapspace.
- freeswap**amount of free swapspace.
- procs**number of current processes.

Errors: None.

See also: Uname ([252](#))

Listing: linuxex/ex64.pp

```
program Example64;  
  
  { Example to demonstrate the SysInfo function }  
  
  Uses Linux;  
  
  Function Mb(L : Longint) : longint;  
  
  begin  
    Mb:=L div (1024*1024);  
  end;  
  
  Var Info : TSysInfo;  
      D,M,Secs,H : longint;  
  
  begin  
    If Not SysInfo(Info) then  
      Halt(1);  
    With Info do  
      begin  
        D:=Uptime div (3600*24);  
        UpTime:=UpTime mod (3600*24);  
        h:=uptime div 3600;  
        uptime:=uptime mod 3600;  
        m:=uptime div 60;  
        secs:=uptime mod 60;  
        Writeln('Uptime : ',d,'days', ' ',h,' hours', ' ',m,' min', ' ',secs,' s. ');  
        Writeln('Loads   : ',Loads[1], '/', Loads[2], '/', Loads[3]);  
        Writeln('Total Ram   : ',Mb(totalram), 'Mb. ');  
        Writeln('Free Ram    : ',Mb(freeram), 'Mb. ');  
        Writeln('Shared Ram  : ',Mb(sharedram), 'Mb. ');  
        Writeln('Buffer Ram  : ',Mb(bufferram), 'Mb. ');  
        Writeln('Total Swap  : ',Mb(totalswap), 'Mb. ');  
        Writeln('Free Swap   : ',Mb(freeswap), 'Mb. ');  
      end;  
    end.  
  end.
```

TCDrain

Declaration: Function TCDrain (Fd:longint) : Boolean;

Description: TCDrain waits until all data to file descriptor Fd is transmitted.

The function returns True if the call was succesfull, False otherwise.

Errors: Errors are reported in LinuxError

See also: termios (2)

TCFlow

Declaration: Function TCFlow (Fd,Act:longint) : Boolean;

Description: TCFlow suspends/resumes transmission or reception of data to or from the file descriptor Fd, depending on the action Act. This can be one of the following pre-defined values:

TCOOFF suspend reception/transmission,
TCOON resume reception/transmission,
TCIOFF transmit a stop character to stop input from the terminal,
TCION transmit start to resume input from the terminal.

The function returns `True` if the call was succesfull, `False` otherwise.

Errors: Errors are reported in `LinuxError`.

See also: `termios (2)`

TCFlush

Declaration: `Function TCFlush (Fd,QSel:longint) : Boolean;`

Description: `TCFlush` discards all data sent or received to/from file descriptor `fd`. `QSel` indicates which queue should be discard. It can be one of the following pre-defined values :

TCIFLUSH input,
TCOFLUSH output,
TCIOFLUSH both input and output.

The function returns `True` if the call was succesfull, `False` otherwise.

Errors: Errors are reported in `LinuxError`.

See also: `termios (2)`

TCGetAttr

Declaration: `Function TCGetAttr (fd:longint;var tios:TermIOS) : Boolean;`

Description: `TCGetAttr` gets the terminal parameters from the terminal referred to by the file descriptor `fd` and returns them in a `TermIOS` structure `tios`. The function returns `True` if the call was succesfull, `False` otherwise.

Errors: Errors are reported in `LinuxError`

See also: `TCSetAttr (250)`, `termios (2)`

Listing: `linuxex/ex55.pp`

Program `Example55;`

uses `Linux;`

{ Program to demonstrate the TCGetAttr/TCSetAttr/CFMakeRaw functions. }

```
procedure ShowTermios(var tios:Termios);
begin
  WriteLn('Input Flags   : $',hexstr(tios.c_iflag,8)+#13);
  WriteLn('Output Flags  : $',hexstr(tios.c_oflag,8));
  WriteLn('Line Flags    : $',hexstr(tios.c_lflag,8));
  WriteLn('Control Flags: $',hexstr(tios.c_cflag,8));
end;

var
```

```
    oldios,
    tios : Termios;
begin
    WriteLn('Old attributes:');
    TCGetAttr(1, tios);
    ShowTermios(tios);
    oldios:=tios;
    WriteLn('Setting raw terminal mode');
    CFMakeRaw(tios);
    TCSetAttr(1, TCSANOW, tios);
    WriteLn('Current attributes:');
    TCGetAttr(1, tios);
    ShowTermios(tios);
    TCSetAttr(1, TCSANOW, oldios);
end.
```

TCGetPGrp

Declaration: Function TCGetPGrp (Fd:longint;var Id:longint) : boolean;

Description: TCGetPGrp returns the process group ID of a foreground process group in Id The function returns True if the call was succesfull, False otherwise

Errors: Errors are reported in LinuxError

See also: termios (2)

TCSendBreak

Declaration: Function TCSendBreak (Fd,Duration:longint) : Boolean;

Description: TCSendBreak Sends zero-valued bits on an asynchrone serial connection decsribed by file-descriptor Fd, for duration Duration. The function returns True if the action was performed successfully, False otherwise.

Errors: Errors are reported in LinuxError.

See also: termios (2)

TCSetAttr

Declaration: Function TCSetAttr (Fd:longint;OptAct:longint;var Tios:TermIOS) : Boolean;

Description: TCSetAttr Sets the terminal parameters you specify in a TermIOS structure Tios for the terminal referred to by the file descriptor Fd. OptAct specifies an optional action when the set need to be done, this could be one of the following pre-defined values:

TCSANOW set immediately.

TCSADRAIN wait for output.

TCSAFLUSH wait for output and discard all input not yet read.

The function Returns True if the call was succesfull, False otherwise.

Errors: Errors are reported in LinuxError.

See also: TCGetAttr ([249](#)), termios (2)

For an example, see TCGetAttr ([249](#)).

TCSetsPGrp

Declaration: `Function TCSetsPGrp (Fd,Id:longint) : boolean;`

Description: `TCSetsPGrp` Sets the Process Group Id to `Id`. The function returns `True` if the call was successful, `False` otherwise.

Errors: Errors are returned in `LinuxError`.

See also: `TCGetPGrp` (250), `termios` (2)

For an example, see `TCGetPGrp` (250).

TTYName

Declaration: `Function TTYName (var f) : String;`

Description: Returns the name of the terminal pointed to by `f`. `f` must be a terminal. `f` can be of type:

1. `longint` for file handles;
2. Text for text variables such as `input` etc.

Errors: Returns an empty string in case of an error. `Linuxerror` may be set to indicate what error occurred, but this is uncertain.

See also: `IsATTY` (226), `IOctl` (225)

TellDir

Declaration: `Function TellDir (p:pdir) : longint;`

Description: `TellDir` returns the current location in the directory structure pointed to by `p`. It returns -1 on failure.

Errors: Errors are returned in `LinuxError`.

See also: `CloseDir` (198), `ReadDir` (236), `SeekDir` (239), `OpenDir` (234), `telldir` (3)

For an example, see `OpenDir` (234).

Umask

Declaration: `Function Umask (Mask : Integer) : Integer;`

Description: Change the file creation mask for the current user to `Mask`. The current mask is returned.

Errors: None

See also: `Chmod` (195), `Umask` (2)

Listing: `linuxex/ex27.pp`

Program `Example27;`

{ Program to demonstrate the Umask function. }

Uses `linux;`

```
begin
  Writeln ( 'Old Umask was : ', Umask(Octal(111)));
  Writeln ( 'New Umask is   : ', Octal(111));
end.
```

Uname

Declaration: Procedure Uname (var unamerec:utsname);

Description: Uname gets the name and configuration of the current LINUX kernel, and returns it in unamerec.

Errors: `LinuxError` is used to report errors.

See also: `GetHostName` (220), `GetDomainName` (217), `uname` (2)

UnLink

Declaration: Function UnLink (Var Path) : Boolean;

Description: `UnLink` decreases the link count on file `Path`. `Path` can be of type `PathStr` or `PChar`. If the link count is zero, the file is removed from the disk. The function returns `True` if the call was successful, `False` if the call failed.

Errors: Errors are returned in `LinuxError`.

sys_eaccess You have no write access right in the directory containing `Path`, or you have no search permission in one of the directory components of `Path`.

sys_eperm The directory containing `pathname` has the sticky-bit set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

sys_enoent A component of the path doesn't exist.

sys_enotdir A directory component of the path is not a directory.

sys_eisdir `Path` refers to a directory.

sys_enomem Insufficient kernel memory.

sys_erofs `Path` is on a read-only filesystem.

See also: `Link` (229), `SymLink` (246), `Unlink` (2)

For an example, see `Link` (229).

Utime

Declaration: Function Utime (path : pathstr; utim : utimbuf) : Boolean;

Description: `Utime` sets the access and modification times of a file. the `utimbuf` record contains 2 fields, `actime`, and `modtime`, both of type `Longint`. They should be filled with an epoch-like time, specifying, respectively, the last access time, and the last modification time. For some filesystem (most notably, FAT), these times are the same.

Errors: Errors are returned in `LinuxError`.

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoent A directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

Other errors may occur, but aren't documented.

See also: [GetEpochTime \(219\)](#), [Chown \(194\)](#), [Access \(189\)](#), [utime \(\(\) 2\)](#)

Listing: linuxex/ex25.pp

Program Example25;

{ Program to demonstrate the UTime function. }

Uses linux;

Var utim : utimbuf;
year, month, day, hour, minute, second : Word;

begin
 { Set access and modification time of executable source }
 GetTime (hour, minute, second);
 GetDate (year, month, day);
 utim . actime := LocalToEpoch (year, month, day, hour, minute, second);
 utim . modtime := utim . actime;
 if not Utime ('ex25.pp', utim) **then**
 writeln ('Call to UTime failed !')
 else
 begin
 Write ('Set access and modification times to : ');
 Write (Hour:2, ':', minute:2, ':', second, ', ');
 WriteLn (Day:2, '/', month:2, '/', year:4);
 end;
end.

WaitPid

Declaration: Function WaitPid (Pid : longint; Status : pointer; Options : Longint)
 : Longint;

Description: WaitPid waits for a child process with process ID Pid to exit. The value of Pid can be one of the following:

Pid < -1 Causes WaitPid to wait for any child process whose process group ID equals the absolute value of pid.

Pid = -1 Causes WaitPid to wait for any child process.

Pid = 0 Causes WaitPid to wait for any child process whose process group ID equals the one of the calling process.

Pid > 0 Causes WaitPid to wait for the child whose process ID equals the value of Pid.

The Options parameter can be used to specify further how WaitPid behaves:

WNOHANG Causes Waitpid to return immediately if no child has exited.

WUNTRACED Causes WaitPid to return also for children which are stopped, but whose status has not yet been reported.

__WCLONE Causes WaitPid also to wait for threads created by the [Clone \(196\)](#) call.

Upon return, it returns the exit status of the process, or -1 in case of failure.

Errors: Errors are returned in LinuxError.

See also: Fork (216), Execve (204), waitpid (2)

For an example, see Fork (216).

WritePort

Declaration: Procedure WritePort (Port : Longint; Value : Byte); Procedure WritePort (Port : Longint; Value : Word); Procedure WritePort (Port : Longint; Value : Longint);

Description: WritePort writes Value – 1 byte, Word or longint – to port Port.

Note: You need permission to write to a port. This permission can be set with root permission with the IOperm call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: IOperm (225), WritePortB (254), WritePortL (254), WritePortW (255), ReadPortB (238), ReadPortL (238), ReadPortW (239)

WritePortB

Declaration: Procedure WritePortB (Port : Longint; Value : Byte); Procedure WritePortB (Port : Longint; Var Buf; Count: longint);

Description: The first form of WritePortB writes 1 byte to port Port. The second form writes Count bytes from Buf to port Port.

Note: You need permission to write to a port. This permission can be set with root permission with the IOperm call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: IOperm (225), WritePort (254), WritePortL (254), WritePortW (255), ReadPortB (238), ReadPortL (238), ReadPortW (239)

WritePortL

Declaration: Procedure WritePortL (Port : Longint; Value : Longint); Procedure WritePortL (Port : Longint; Var Buf; Count: longint);

Description: The first form of WritePortB writes 1 byte to port Port. The second form writes Count bytes from Buf to port Port.

Note: You need permission to write to a port. This permission can be set with root permission with the IOperm call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: IOperm (225), WritePort (254), WritePortB (254), WritePortW (255), ReadPortB (238), ReadPortL (238), ReadPortW (239)

WritePortW

Declaration: `Procedure WritePortW (Port : Longint; Var Buf; Count: longint); Procedure WritePortW (Port : Longint; Value : Word);`

Description: The first form of WritePortB writes 1 byte to port Port. The second form writes Count bytes from Buf to port Port.

Note: You need permission to write to a port. This permission can be set with root permission with the IOperm call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: IOperm ([225](#)), WritePort ([254](#)), WritePortL ([254](#)), WritePortB ([254](#)), ReadPortB ([238](#)), ReadPortL ([238](#)), ReadPortW ([239](#))

Chapter 13

The MATH unit

This chapter describes the `math` unit. The `math` unit was initially written by Florian Klaempfl. It provides mathematical functions which aren't covered by the system unit.

This chapter starts out with a definition of all types and constants that are defined, after which an overview is presented of the available functions, grouped by category, and the last part contains a complete explanation of each function.

The following things must be taken into account when using this unit:

1. This unit is compiled in Object Pascal mode so all `integers` are 32 bit.
2. Some overloaded functions exist for data arrays of integers and floats. When using the address operator (`@`) to pass an array of data to such a function, make sure the address is typecasted to the right type, or turn on the 'typed address operator' feature. failing to do so, will cause the compiler not be able to decide which function you want to call.

13.1 Constants and types

The following types are defined in the `math` unit:

```
Type
  Float = Extended;
  PFloat = ^Float
```

All calculations are done with the `Float` type. This allows to recompile the unit with a different float type to obtain a desired precision. The pointer type is used in functions that accept an array of values of arbitrary length.

```
Type
  TPaymentTime = (PTEndOfPeriod, PTStartOfPeriod);
```

`TPaymentTime` is used in the financial calculations.

```
Type
  EInvalidArgument = Class(EMathError);
```

The `EInvalidArgument` exception is used to report invalid arguments.

13.2 Function list by category

What follows is a listing of the available functions, grouped by category. For each function there is a reference to the page where you can find the function.

Min/max determination

Functions to determine the minimum or maximum of numbers:

Name	Description	Page
max	Maximum of 2 values	270
maxIntValue	Maximum of an array of integer values	271
maxvalue	Maximum of an array of values	271
min	Minimum of 2 values	273
minIntValue	Minimum of an array of integer values	274
minvalue	Minimum of an array of values	275

Angle conversion

Name	Description	Page
cycleto rad	convert cycles to radians	263
degtograd	convert degrees to grads	264
degtorad	convert degrees to radians	264
gradtodeg	convert grads to degrees	266
gradtorad	convert grads to radians	266
radto cycle	convert radians to cycles	279
radtodeg	convert radians to degrees	279
radto grad	convert radians to grads	280

Trigonometric functions

Name	Description	Page
arccos	calculate reverse cosine	259
arcsin	calculate reverse sine	260
arctan2	calculate reverse tangent	261
cotan	calculate cotangent	263
sincos	calculate sine and cosine	281
tan	calculate tangent	284

Hyperbolic functions

Name	Description	Page
arcosh	calculate reverse hyperbolic cosine	259
arsinh	calculate reverse hyperbolic sine	261
artanh	calculate reverse hyperbolic tangent	262
cosh	calculate hyperbolic cosine	263
sinh	calculate hyperbolic sine	281
tanh	calculate hyperbolic tangent	285

Exponential and logarithmic functions

Name	Description	Page
intpower	Raise float to integer power	267
ldexp	Calculate $2^p x$	268
lnxp1	calculate $\log(x+1)$	268
log10	calculate 10-base log	269
log2	calculate 2-base log	269
logn	calculate N-base log	270
power	raise float to arbitrary power	278

Number converting

Name	Description	Page
ceil	Round to infinity	262
floor	Round to minus infinity	265
frexp	Return mantissa and exponent	265

Statistical functions

Name	Description	Page
mean	Mean of values	272
meanandstddev	Mean and standard deviation of values	273
momentskewkurtosis	Moments, skew and kurtosis	275
popnstddev	Population standard deviation	277
popnvariance	Population variance	277
randg	Gaussian distributed random value	280
stddev	Standard deviation	282
sum	Sum of values	282
sumofsquares	Sum of squared values	283

<code>sumsandsquares</code>	Sum of values and squared values	284
<code>totalvariance</code>	Total variance of values	285
<code>variance</code>	variance of values	286

Geometrical functions

Name	Description	Page
<code>hypot</code>	Hypotenuse of triangle	267
<code>norm</code>	Euclidian norm	276

13.3 Functions and Procedures

arccos

Declaration: `Function arccos(x : float) : float;`

Description: `Arccos` returns the inverse cosine of its argument `x`. The argument `x` should lie between -1 and 1 (borders included).

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `arcsin` ([260](#)), `arcosh` ([259](#)), `arsinh` ([261](#)), `artanh` ([262](#))

Listing: `mathex/ex1.pp`

Program `Example1;`

{ Program to demonstrate the arccos function. }

Uses `math;`

Procedure `WriteRadDeg(X : float);`

begin

`WriteLn(X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees.')`

`end;`

begin

`WriteRadDeg (arccos (1));`

`WriteRadDeg (arccos (sqrt(3)/2));`

`WriteRadDeg (arccos (sqrt(2)/2));`

`WriteRadDeg (arccos (1/2));`

`WriteRadDeg (arccos (0));`

`WriteRadDeg (arccos (-1));`

`end.`

arcosh

Declaration: `Function arcosh(x : float) : float; Function arccosh(x : float) : float;`

Description: `Arcosh` returns the inverse hyperbolic cosine of its argument `x`. The argument `x` should be larger than 1.

The `arccosh` variant of this function is supplied for Delphi compatibility.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `cosh` ([263](#)), `sinh` ([281](#)), `arcsin` ([260](#)), `arsinh` ([261](#)), `artanh` ([262](#)), `tanh` ([285](#))

Listing: `mathex/ex3.pp`

Program `Example3;`

{ Program to demonstrate the arcosh function. }

Uses `math;`

begin

WriteLn(`arcosh(1)`);

WriteLn(`arcosh(2)`);

end.

arcsin

Declaration: `Function arcsin(x : float) : float;`

Description: `Arcsin` returns the inverse sine of its argument `x`. The argument `x` should lie between -1 and 1.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `arccos` ([259](#)), `arcosh` ([259](#)), `arsinh` ([261](#)), `artanh` ([262](#))

Listing: `mathex/ex2.pp`

Program `Example1;`

{ Program to demonstrate the arcsin function. }

Uses `math;`

Procedure `WriteRadDeg(X : float);`

begin

WriteLn(`X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees.'`)

end;

begin

WriteRadDeg (`arcsin(1)`);

WriteRadDeg (`arcsin(sqrt(3)/2)`);

WriteRadDeg (`arcsin(sqrt(2)/2)`);

WriteRadDeg (`arcsin(1/2)`);

WriteRadDeg (`arcsin(0)`);

WriteRadDeg (`arcsin(-1)`);

end.

arctan2

Declaration: `Function arctan2(x,y : float) : float;`

Description: `arctan2` calculates `arctan(y/x)`, and returns an angle in the correct quadrant. The returned angle will be in the range $-\pi$ to π radians. The values of `x` and `y` must be between -2^{64} and 2^{64} , moreover `x` should be different from zero.

On Intel systems this function is implemented with the native intel `fpatan` instruction.

Errors: If `x` is zero, an overflow error will occur.

See also: `arccos` ([259](#)), `arcosh` ([259](#)), `arsinh` ([261](#)), `artanh` ([262](#))

Listing: `mathex/ex6.pp`

Program `Example6;`

{ Program to demonstrate the arctan2 function. }

Uses `math;`

Procedure `WriteRadDeg(X : float);`

begin

`WriteLn(X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees.')`

end;

begin

`WriteRadDeg (arctan2 (1,1));`

end.

arsinh

Declaration: `Function arsinh(x : float) : float; Function arcsinh(x : float) : float;`

Description: `arsinh` returns the inverse hyperbolic sine of its argument `x`.

The `arcsinh` variant of this function is supplied for Delphi compatibility.

Errors: None.

See also: `arcosh` ([259](#)), `arccos` ([259](#)), `arcsin` ([260](#)), `artanh` ([262](#))

Listing: `mathex/ex4.pp`

Program `Example4;`

{ Program to demonstrate the arsinh function. }

Uses `math;`

begin

`WriteLn (arsinh (0));`

`WriteLn (arsinh (1));`

end.

artanh

Declaration: `Function artanh(x : float) : float; Function arctanh(x : float) : float;`

Description: `artanh` returns the inverse hyperbolic tangent of its argument `x`, where `x` should lie in the interval `[-1,1]`, borders included.

The `arctanh` variant of this function is supplied for Delphi compatibility.

Errors: In case `x` is not in the interval `[-1,1]`, an `EInvalidArgument` exception is raised.

See also: `arcosh` ([259](#)), `arccos` ([259](#)), `arcsin` ([260](#)), `artanh` ([262](#))

Errors:

See also:

Listing: `mathex/ex5.pp`

Program `Example5;`

{ Program to demonstrate the artanh function. }

Uses `math;`

begin

Writeln(`artanh(0)`);

Writeln(`artanh(0.5)`);

end.

ceil

Declaration: `Function ceil(x : float) : longint;`

Description: `Ceil` returns the lowest integer number greater than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If the absolute value of `x` is larger than `maxint`, an overflow error will occur.

See also: `floor` ([265](#))

Listing: `mathex/ex7.pp`

Program `Example7;`

{ Program to demonstrate the Ceil function. }

Uses `math;`

begin

Writeln(`Ceil(-3.7)`); // should be -3

Writeln(`Ceil(3.7)`); // should be 4

Writeln(`Ceil(-4.0)`); // should be -4

end.

cosh

Declaration: `Function cosh(x : float) : float;`

Description: Cosh returns the hyperbolic cosine of its argument x.

Errors: None.

See also: [arcosh \(259\)](#), [sinh \(281\)](#), [arsinh \(261\)](#)

Listing: mathex/ex8.pp

Program Example8;

{ Program to demonstrate the cosh function . }

Uses math;

begin
 Writeln (Cosh (0));
 Writeln (Cosh (1));
end.

cotan

Declaration: `Function cotan(x : float) : float;`

Description: Cotan returns the cotangent of its argument x. x should be different from zero.

Errors: If x is zero then a overflow error will occur.

See also: [tanh \(285\)](#)

Listing: mathex/ex9.pp

Program Example9;

{ Program to demonstrate the cotan function . }

Uses math;

begin
 writeln (cotan (pi / 2));
 Writeln (cotan (pi / 3));
 Writeln (cotan (pi / 4));
end.

cycletorad

Declaration: `Function cycletorad(cycle : float) : float;`

Description: Cycletorad transforms its argument cycle (an angle expressed in cycles) to radians. (1 cycle is 2π radians).

Errors: None.

See also: [degtograd \(264\)](#), [degtorad \(264\)](#), [radto deg \(279\)](#), [radtograd \(280\)](#), [radto cycle \(279\)](#)

Listing: mathex/ex10.pp

Program Example10;*{ Program to demonstrate the cycletorad function. }***Uses** math;**begin****writeln**(**cos**(cycletorad(1/6))); // Should print 1/2**writeln**(**cos**(cycletorad(1/8))); // should be **sqrt**(2)/2**end.**

degtograd

Declaration: Function degtograd(deg : float) : float;

Description: Degtograd transforms it's argument deg (an angle in degrees) to grads.
(90 degrees is 100 grad.)

Errors: None.

See also: cycletorad ([263](#)), degtorad ([264](#)), radtodeg ([279](#)), radtograd ([280](#)), radtcycle ([279](#))**Listing:** mathex/ex11.pp

Program Example11;*{ Program to demonstrate the degtograd function. }***Uses** math;**begin****writeln**(degtograd(90));**writeln**(degtograd(180));**writeln**(degtograd(270))**end.**

degtorad

Declaration: Function degtorad(deg : float) : float;

Description: Degtorad converts it's argument deg (an angle in degrees) to radians.
(pi radians is 180 degrees)

Errors: None.

See also: cycletorad ([263](#)), degtograd ([264](#)), radtodeg ([279](#)), radtograd ([280](#)), radtcycle ([279](#))**Listing:** mathex/ex12.pp

Program Example12;*{ Program to demonstrate the degtorad function. }***Uses** math;

```
begin
  writeln (degtorad(45));
  writeln (degtorad(90));
  writeln (degtorad(180));
  writeln (degtorad(270));
  writeln (degtorad(360));
end.
```

floor

Declaration: `Function floor(x : float) : longint;`

Description: `Floor` returns the largest integer smaller than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If `x` is larger than `maxint`, an overflow will occur.

See also: `ceil` ([262](#))

Listing: `mathex/ex13.pp`

Program `Example13;`

{ Program to demonstrate the floor function. }

Uses `math;`

```
begin
  Writeln (Ceil (-3.7)); // should be -4
  Writeln (Ceil (3.7));  // should be 3
  Writeln (Ceil (-4.0)); // should be -4
end.
```

frexp

Declaration: `Procedure frexp(x : float; var mantissa : float; var exponent : integer);`

Description: `Frexp` returns the mantissa and exponent of its argument `x` in mantissa and exponent.

Errors: None

See also:

Listing: `mathex/ex14.pp`

Program `Example14;`

{ Program to demonstrate the frexp function. }

Uses `math;`

Procedure `dofrexp(Const X : extended);`

```
var man : extended;
    exp : integer;
```

begin

```
man:=0;  
exp:=0;  
frexp(x,man,exp);  
write(x, ' has ');  
Writeln('mantissa ',man,' and exponent ',exp);  
end;
```

```
begin  
  //  dofrex(1.00);  
  dofrex(1.02e-1);  
  dofrex(1.03e-2);  
  dofrex(1.02e1);  
  dofrex(1.03e2);  
end.
```

gradtodeg

Declaration: Function gradtodeg(grad : float) : float;

Description: Gradtodeg converts its argument grad (an angle in grads) to degrees.
(100 grad is 90 degrees)

Errors: None.

See also: cycletorad ([263](#)), degtograd ([264](#)), radtodeg ([279](#)), radtograd ([280](#)), radtcycle ([279](#)), gradtorad ([266](#))

Listing: mathex/ex15.pp

Program Example15;

```
{ Program to demonstrate the gradtodeg function. }
```

Uses math;

```
begin  
  writeln(gradtodeg(100));  
  writeln(gradtodeg(200));  
  writeln(gradtodeg(300));  
end.
```

gradtorad

Declaration: Function gradtorad(grad : float) : float;

Description: Gradtorad converts its argument grad (an angle in grads) to radians.
(200 grad is pi degrees).

Errors: None.

See also: cycletorad ([263](#)), degtograd ([264](#)), radtodeg ([279](#)), radtograd ([280](#)), radtcycle ([279](#)), gradtodeg ([266](#))

Listing: mathex/ex16.pp

Program Example16;

{ Program to demonstrate the gradtorad function. }

Uses math;

begin

writeln(gradtorad(100));

writeln(gradtorad(200));

writeln(gradtorad(300));

end.

hypot

Declaration: Function hypot(x,y : float) : float;

Description: Hypot returns the hypotenuse of the triangle where the sides adjacent to the square angle have lengths x and y.

The function uses Pythagoras' rule for this.

Errors: None.

See also:

Listing: mathex/ex17.pp

Program Example17;

{ Program to demonstrate the hypot function. }

Uses math;

begin

Writeln(hypot(3,4)); // should be 5

end.

intpower

Declaration: Function intpower(base : float;exponent : longint) : float;

Description: Intpower returns base to the power exponent, where exponent is an integer value.

Errors: If base is zero and the exponent is negative, then an overflow error will occur.

See also: power ([278](#))

Listing: mathex/ex18.pp

Program Example18;

{ Program to demonstrate the intpower function. }

Uses math;

Procedure DoIntpower (X : extended; Pow : Integer);

```
begin
  writeln(X:8:4, '^', Pow:2, ' = ', intpower(X,pow):8:4);
end;

begin
  dointpower(0.0,0);
  dointpower(1.0,0);
  dointpower(2.0,5);
  dointpower(4.0,3);
  dointpower(2.0,-1);
  dointpower(2.0,-2);
  dointpower(-2.0,4);
  dointpower(-4.0,3);
end.
```

Idexp

Declaration: Function `ldexp(x : float;p : longint) : float;`

Description: `Ldexp` returns $2^p x$.

Errors: None.

See also: `lnxp1` ([268](#)), `log10` ([269](#)), `log2` ([269](#)), `logn` ([270](#))

Listing: mathex/ex19.pp

Program Example19;

{ Program to demonstrate the Idexp function. }

Uses math;

```
begin
  writeln(Idexp(2,4):8:4);
  writeln(Idexp(0.5,3):8:4);
end.
```

lnxp1

Declaration: Function `lnxp1(x : float) : float;`

Description: `lnxp1` returns the natural logarithm of $1+x$. The result is more precise for small values of x . x should be larger than -1 .

Errors: If $x \leq -1$ then an `EInvalidArgument` exception will be raised.

See also: `ldexp` ([268](#)), `log10` ([269](#)), `log2` ([269](#)), `logn` ([270](#))

Listing: mathex/ex20.pp

Program Example20;

{ Program to demonstrate the lnxp1 function. }

Uses math;

```
begin
  writeln(lnxp1(0));
  writeln(lnxp1(0.5));
  writeln(lnxp1(1));
end.
```

log10

Declaration: `Function log10(x : float) : float;`

Description: `Log10` returns the 10-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` ([268](#)), `lnxp1` ([268](#)), `log2` ([269](#)), `logn` ([270](#))

Listing: mathex/ex21.pp

Program Example21;

{ Program to demonstrate the log10 function. }

Uses math;

```
begin
  Writeln(Log10(10):8:4);
  Writeln(Log10(100):8:4);
  Writeln(Log10(1000):8:4);
  Writeln(Log10(1):8:4);
  Writeln(Log10(0.1):8:4);
  Writeln(Log10(0.01):8:4);
  Writeln(Log10(0.001):8:4);
end.
```

log2

Declaration: `Function log2(x : float) : float;`

Description: `Log2` returns the 2-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` ([268](#)), `lnxp1` ([268](#)), `log10` ([269](#)), `logn` ([270](#))

Listing: mathex/ex22.pp

Program Example22;

{ Program to demonstrate the log2 function. }

Uses math;

```
begin
  Writeln(Log2(2):8:4);
  Writeln(Log2(4):8:4);
  Writeln(Log2(8):8:4);
  Writeln(Log2(1):8:4);
```

```
Writeln(Log2(0.5):8:4);
Writeln(Log2(0.25):8:4);
Writeln(Log2(0.125):8:4);
end.
```

logn

Declaration: `Function logn(n,x : float) : float;`

Description: Logn returns the n-base logarithm of X.

Errors: If x is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: Idexp ([268](#)), Inxp1 ([268](#)), log10 ([269](#)), log2 ([269](#))

Listing: mathex/ex23.pp

Program Example23;

{ Program to demonstrate the logn function. }

Uses math;

begin

```
Writeln(Logn(3,4):8:4);
Writeln(Logn(2,4):8:4);
Writeln(Logn(6,9):8:4);
Writeln(Logn(exp(1),exp(1)):8:4);
Writeln(Logn(0.5,1):8:4);
Writeln(Logn(0.25,3):8:4);
Writeln(Logn(0.125,5):8:4);
```

end.

max

Declaration: `Function max(Int1,Int2:Cardinal):Cardinal; Function max(Int1,Int2:Integer):Integer;`

Description: Max returns the maximum of Int1 and Int2.

Errors: None.

See also: min ([273](#)), maxIntValue ([271](#)), maxvalue ([271](#))

Listing: mathex/ex24.pp

Program Example24;

{ Program to demonstrate the max function. }

Uses math;

Var

A,B : Cardinal;

begin

```
A:=1;b:=2;
writeln(max(a,b));
```

end.

maxIntValue

Declaration: `function MaxIntValue(const Data: array of Integer): Integer;`

Description: `MaxIntValue` returns the largest integer out of the `Data` array.

This function is provided for Delphicompatibility, use the `maxvalue` (271) function instead.

Errors: None.

See also: `maxvalue` (271), `minvalue` (275), `minIntValue` (274)

Listing: `mathex/ex25.pp`

Program `Example25;`

{ Program to demonstrate the MaxIntValue function. }

{ Make sore integer is 32 bit }
{ \$mode objfpc }

Uses `math;`

Type

`TExArray = Array[1..100] of Integer;`

Var

`I : Integer;`
`ExArray : TExArray;`

begin

`Randomize;`
`for I:=1 to 100 do`
`ExArray[i]:=Random(I)-Random(100);`
`WriteLn(MaxIntValue(ExArray));`

end.

maxvalue

Declaration: `Function maxvalue(const data : array of float) : float; Function maxvalue(const data : array of Integer) : Integer; Function maxvalue(const data : PFloat; Const N : Integer) : float; Function maxvalue(const data : PInteger; Const N : Integer) : Integer;`

Description: `Maxvalue` returns the largest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: `maxIntValue` (271), `minvalue` (275), `minIntValue` (274)

Listing: `mathex/ex26.pp`

Program `Example26;`

{ Program to demonstrate the MaxValue function. }

{ Make sore integer is 32 bit }

```
{ $mode objfpc }
```

```
Uses math;
```

Type

```
TExFloatArray = Array[1..100] of Float;
TExIntArray = Array[1..100] of Integer;
```

Var

```
l : Integer;
ExFloatArray : TExFloatArray;
ExIntArray : TExIntArray;
AFloatArray : PFloat;
AIntArray : PInteger;
```

begin

```
Randomize;
AFloatArray := @ExFloatArray[1];
AIntArray := @ExIntArray[1];
for l:=1 to 100 do
  ExFloatArray[l] := (Random-Random)*100;
for l:=1 to 100 do
  ExIntArray[l] := Random(l)-Random(100);
Writeln('Max Float      : ', MaxValue(ExFloatArray):8:4);
Writeln('Max Float (b) : ', MaxValue(AFloatArray,100):8:4);
Writeln('Max Integer    : ', MaxValue(ExIntArray):8);
Writeln('Max Integer (b) : ', MaxValue(AIntArray,100):8);
```

```
end.
```

mean

Declaration: Function mean(const data : array of float) : float; Function mean(const data : PFloat; Const N : longint) : float;

Description: Mean returns the average value of data.

The second form accepts a pointer to an array of N values.

Errors: None.

See also: [meanandstddev \(273\)](#), [momentskewkurtosis \(275\)](#), [sum \(282\)](#)

Listing: mathex/ex27.pp

Program Example27;

```
{ Program to demonstrate the Mean function. }
```

```
Uses math;
```

Type

```
TExArray = Array[1..100] of Float;
```

Var

```
l : Integer;
ExArray : TExArray;
```

begin

```
Randomize;
```

```
for I:=1 to 100 do
  ExArray[i]:=(Random-Random)*100;
  Writeln('Max      : ',MaxValue(ExArray):8:4);
  Writeln('Min      : ',MinValue(ExArray):8:4);
  Writeln('Mean     : ',Mean(ExArray):8:4);
  Writeln('Mean (b) : ',Mean(@ExArray[1],100):8:4);
end.
```

meanandstddev

Declaration: Procedure meanandstddev(const data : array of float; var mean,stddev : float); procedure meanandstddev(const data : PFloat; Const N : Longint;var mean,stddev : float);

Description: meanandstddev calculates the mean and standard deviation of data and returns the result in mean and stddev, respectively. Stddev is zero if there is only one value.

The second form accepts a pointer to an array of N values.

Errors: None.

See also: mean ([272](#)),sum ([282](#)), sumofsquares ([283](#)), momentskewkurtosis ([275](#))

Listing: mathex/ex28.pp

Program Example28;

{ Program to demonstrate the Meanandstddev function. }

Uses math;

Type

TExArray = **Array**[1..100] **of** Extended;

Var

I : Integer;
ExArray : TExArray;
Mean,stddev : Extended;

begin

```
  Randomize;
  for I:=1 to 100 do
    ExArray[i]:=(Random-Random)*100;
  MeanAndStdDev(ExArray,Mean,StdDev);
  Writeln('Mean      : ',Mean:8:4);
  Writeln('StdDev     : ',StdDev:8:4);
  MeanAndStdDev(@ExArray[1],100,Mean,StdDev);
  Writeln('Mean (b) : ',Mean:8:4);
  Writeln('StdDev (b) : ',StdDev:8:4);
end.
```

min

Declaration: Function min(Int1,Int2:Cardinal):Cardinal; Function min(Int1,Int2:Integer):Integer;

Description: min returns the smallest value of Int1 and Int2;

Errors: None.

See also: [max \(270\)](#)

Listing: mathex/ex29.pp

Program Example29;

{ Program to demonstrate the min function . }

Uses math;

Var

A,B : Cardinal;

begin

A:=1;b:=2;

writeln(min(a,b));

end.

minIntValue

Declaration: Function minIntValue(const Data: array of Integer): Integer;

Description: MinIntvalue returns the smallest value in the Data array.

This function is provided for Delphicompatibility, use minvalue instead.

Errors: None

See also: [minvalue \(275\)](#), [maxIntValue \(271\)](#), [maxvalue \(271\)](#)

Listing: mathex/ex30.pp

Program Example30;

{ Program to demonstrate the MinIntValue function . }

{ Make sore integer is 32 bit }

{ \$mode objfpc }

Uses math;

Type

TExArray = **Array**[1..100] **of** Integer;

Var

I : Integer;

ExArray : TExArray;

begin

Randomize;

for I:=1 **to** 100 **do**

ExArray[I]:=Random(I)-Random(100);

Writeln(MinIntValue(ExArray));

end.

minvalue

Declaration: Function minvalue(const data : array of float) : float; Function minvalue(const data : array of Integer) : Integer; Function minvalue(const data : PFloat; Const N : Integer) : float; Function minvalue(const data : PInteger; Const N : Integer) : Integer;

Description: Minvalue returns the smallest value in the data array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of N integer or float values.

Errors: None.

See also: [maxIntValue \(271\)](#), [maxvalue \(271\)](#), [minIntValue \(274\)](#)

Listing: mathex/ex31.pp

Program Example26;

```
{ Program to demonstrate the MinValue function. }

{ Make sure integer is 32 bit }
{$mode objfpc}
```

Uses math;

Type

```
TExFloatArray = Array[1..100] of Float;
TExIntArray = Array[1..100] of Integer;
```

Var

```
I : Integer;
ExFloatArray : TExFloatArray;
AFloatArray : PFloat;
ExIntArray : TExIntArray;
AIntArray : PInteger;
```

begin

```
Randomize;
AFloatArray := @ExFloatArray[0];
AIntArray := @ExIntArray[0];
for I:=1 to 100 do
  ExFloatArray[I] := (Random-Random)*100;
for I:=1 to 100 do
  ExIntArray[I] := Random(I)-Random(100);
Writeln('Min Float      : ', MinValue(ExFloatArray):8:4);
Writeln('Min Float    (b) : ', MinValue(AFloatArray,100):8:4);
Writeln('Min Integer   : ', MinValue(ExIntArray):8);
Writeln('Min Integer (b) : ', MinValue(AIntArray,100):8);
```

end.

momentskewkurtosis

Declaration: procedure momentskewkurtosis(const data : array of float; var m1,m2,m3,m4,skew,kurtosis : float); procedure momentskewkurtosis(const data : PFloat; Const N : Integer; var m1,m2,m3,m4,skew,kurtosis : float);

Description: `momentskewkurtosis` calculates the 4 first moments of the distribution of values in data and returns them in `m1,m2,m3` and `m4`, as well as the skew and kurtosis.

Errors: None.

See also: `mean` (272), `meanandstddev` (273)

Listing: `mathex/ex32.pp`

Program `Example32`;

{ Program to demonstrate the momentskewkurtosis function. }

Uses `math`;

Var

`DistArray : Array[1..1000] of float;`
`l : longint;`
`m1,m2,m3,m4,skew,kurtosis : float;`

begin

`randomize;`
`for l:=1 to 1000 do`
 `distarray[l]:=random;`
`momentskewkurtosis(DistArray,m1,m2,m3,m4,skew,kurtosis);`

`Writeln('1st moment : ',m1:8:6);`
`Writeln('2nd moment : ',m2:8:6);`
`Writeln('3rd moment : ',m3:8:6);`
`Writeln('4th moment : ',m4:8:6);`
`Writeln('Skew : ',skew:8:6);`
`Writeln('kurtosis : ',kurtosis:8:6);`

end.

norm

Declaration: `Function norm(const data : array of float) : float;` `Function norm(const data : PFloat; Const N : Integer) : float;`

Description: `Norm` calculates the Euclidian norm of the array of data. This equals `sqrt(sumofsquares(data))`.

The second form accepts a pointer to an array of N values.

Errors: None.

See also: `sumofsquares` (283)

Listing: `mathex/ex33.pp`

Program `Example33`;

{ Program to demonstrate the norm function. }

Uses `math`;

Type

`TVector = Array[1..10] of Float;`

Var

```

AVector : Tvector;
l : longint;

begin
  for l:=1 to 10 do
    Avector[l]:=Random;
  Writeln(Norm(AVector));
end.

```

popnstddev

Declaration: Function popnstddev(const data : array of float) : float; Function popnstddev(const data : PFloat; Const N : Integer) : float;

Description: Popnstddev returns the square root of the population variance of the values in the Data array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of N values.

Errors: None.

See also: popnvariance (277), mean (272), meanandstddev (273), stddev (282), momentskewkurtosis (275)

Listing: mathex/ex35.pp

Program Example35;

{ Program to demonstrate the PopnStdDev function. }

Uses Math;

Type

TExArray = **Array**[1..100] **of** Float;

Var

l : Integer;
ExArray : TExArray;

begin

```

  Randomize;
  for l:=1 to 100 do
    ExArray[l]:=(Random-Random)*100;
  Writeln('Max           : ',MaxValue(ExArray):8:4);
  Writeln('Min           : ',MinValue(ExArray):8:4);
  Writeln('Pop. stddev.    : ',PopnStdDev(ExArray):8:4);
  Writeln('Pop. stddev. (b) : ',PopnStdDev(@ExArray[1],100):8:4);
end.

```

popnvariance

Declaration: Function popnvariance(const data : array of float) : float; Function popnvariance(const data : PFloat; Const N : Integer) : float;

Description: Popnvariance returns the square root of the population variance of the values in the Data array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of N values.

Errors: None.

See also: [popnstddev \(277\)](#), [mean \(272\)](#), [meanandstddev \(273\)](#), [stddev \(282\)](#), [momentskewkurtosis \(275\)](#)

Listing: mathex/ex36.pp

Program Example36;

{ Program to demonstrate the PopnVariance function . }

Uses math;

Type

TExArray = **Array**[1..100] of Float;

Var

I : Integer;

ExArray : TExArray;

begin

Randomize;

for I:=1 **to** 100 **do**

ExArray[I]:= (Random-~~Random~~)*100;

WriteLn('Max : ', MaxValue(ExArray):8:4);

WriteLn('Min : ', MinValue(ExArray):8:4);

WriteLn('Pop. var. : ', PopnVariance(ExArray):8:4);

WriteLn('Pop. var. (b) : ', PopnVariance(@ExArray[1],100):8:4);

end.

power

Declaration: `Function power(base,exponent : float) : float;`

Description: `power` raises `base` to the power `power`. This is equivalent to `exp(power*ln(base))`. Therefore `base` should be non-negative.

Errors: None.

See also: [intpower \(267\)](#)

Listing: mathex/ex34.pp

Program Example34;

{ Program to demonstrate the power function . }

Uses Math;

procedure dopower(x,y : float);

begin

writeLn(x:8:6, '^', y:8:6, ' = ', power(x,y):8:6)

end;

begin

dopower(2,2);

dopower(2,-2);


```
dopower(2,0.0);  
end.
```

radtocycle

Declaration: `Function radtocycle(rad : float) : float;`

Description: `Radtocycle` converts its argument `rad` (an angle expressed in radians) to an angle in cycles.
(1 cycle equals 2 pi radians)

Errors: None.

See also: `degtograd` ([264](#)), `degtorad` ([264](#)), `radtodeg` ([279](#)), `radtograd` ([280](#)), `cycletorad` ([263](#))

Listing: `mathex/ex37.pp`

Program `Example37;`

{ Program to demonstrate the radtocycle function. }

Uses `math;`

begin

`writeln(radtocycle(2*pi):8:6);`

`writeln(radtocycle(pi):8:6);`

`writeln(radtocycle(pi/2):8:6);`

end.

radtodeg

Declaration: `Function radtodeg(rad : float) : float;`

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in degrees.
(180 degrees equals pi radians)

Errors: None.

See also: `degtograd` ([264](#)), `degtorad` ([264](#)), `radtocycle` ([279](#)), `radtograd` ([280](#)), `cycletorad` ([263](#))

Listing: `mathex/ex38.pp`

Program `Example38;`

{ Program to demonstrate the radtodeg function. }

Uses `math;`

begin

`writeln(radtodeg(2*pi):8:6);`

`writeln(radtodeg(pi):8:6);`

`writeln(radtodeg(pi/2):8:6);`

end.

radtograd

Declaration: `Function radtograd(rad : float) : float;`

Description: `RadtoDeg` converts its argument `rad` (an angle expressed in radians) to an angle in grads.
(200 grads equals pi radians)

Errors: None.

See also: `degtograd` ([264](#)), `degtorad` ([264](#)), `radtoCycle` ([279](#)), `radtoDeg` ([279](#)), `cycletorad` ([263](#))

Listing: `mathex/ex39.pp`

Program `Example39;`

{ Program to demonstrate the radtograd function. }

Uses `math;`

begin

`writeln(radtograd(2*pi):8:6);`

`writeln(radtograd(pi):8:6);`

`writeln(radtograd(pi/2):8:6);`

end.

randg

Declaration: `Function randg(mean,stddev : float) : float;`

Description: `randg` returns a random number which - when produced in large quantities - has a Gaussian distribution with mean `mean` and standarddeviation `stddev`.

Errors: None.

See also: `mean` ([272](#)), `stddev` ([282](#)), `meanandstddev` ([273](#))

Listing: `mathex/ex40.pp`

Program `Example40;`

{ Program to demonstrate the randg function. }

Uses `Math;`

Type

`TExArray = Array[1..10000] of Float;`

Var

`I : Integer;`

`ExArray : TExArray;`

`Mean,stddev : Float;`

begin

`Randomize;`

`for I:=1 to 10000 do`

`ExArray[I]:=Randg(1,0.2);`

`MeanAndStdDev(ExArray,Mean,StdDev);`

`Writeln('Mean : ',Mean:8:4);`

`Writeln('StdDev : ',StdDev:8:4);`

end.

sincos

Declaration: `Procedure sincos(theta : float; var sinus, cosinus : float);`

Description: `Sincos` calculates the sine and cosine of the angle `theta`, and returns the result in `sinus` and `cosinus`.

On Intel hardware, This calculation will be faster than making 2 calls to calculate the sine and cosine separately.

Errors: None.

See also: [arcsin \(260\)](#), [arccos \(259\)](#).

Listing: `mathex/ex41.pp`

Program `Example41;`

{ Program to demonstrate the sincos function. }

Uses `math;`

Procedure `dosincos(Angle : Float);`

Var

`Sine, Cosine : Float;`

begin

`sincos(angle, sine, cosine);`
Write (`' Angle : ', Angle:8:6`);
Write (`' Sine : ', sine:8:6`);
Write (`' Cosine : ', cosine:8:6`);

end;

begin

`dosincos(pi);`
`dosincos(pi/2);`
`dosincos(pi/3);`
`dosincos(pi/4);`
`dosincos(pi/6);`

end.

sinh

Declaration: `Function sinh(x : float) : float;`

Description: `Sinh` returns the hyperbolic sine of its argument `x`.

Errors:

See also: [cosh \(263\)](#), [arsinh \(261\)](#), [tanh \(285\)](#), [artanh \(262\)](#)

Listing: `mathex/ex42.pp`

Program `Example42;`

{ Program to demonstrate the sinh function. }

Uses `math;`

```
begin
  writeln(sinh(0));
  writeln(sinh(1));
  writeln(sinh(-1));
end.
```

stddev

Declaration: `Function stddev(const data : array of float) : float; Function stddev(const data : PFloat; Const N : Integer) : float;`

Description: Stddev returns the standard deviation of the values in Data. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: [mean \(272\)](#), [meanandstddev \(273\)](#), [variance \(286\)](#), [totalvariance \(285\)](#)

Listing: mathex/ex43.pp

Program Example40;

{ Program to demonstrate the stddev function. }

Uses Math;

Type

TExArray = **Array**[1..10000] of Float;

Var

I : Integer;

ExArray : TExArray;

begin

Randomize;

for I:=1 **to** 10000 **do**

ExArray[I]:=Randg(1,0.2);

Writeln('StdDev : ',StdDev(ExArray):8:4);

Writeln('StdDev (b) : ',StdDev(@ExArray[0],10000):8:4);

end.

sum

Declaration: `Function sum(const data : array of float) : float; Function sum(const data : PFloat; Const N : Integer) : float;`

Description: Sum returns the sum of the values in the data array.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: [sumofsquares \(283\)](#), [sumsandsquares \(284\)](#), [totalvariance \(285\)](#), [variance \(286\)](#)

Listing: mathex/ex44.pp

```
Program Example44;

{ Program to demonstrate the Sum function. }

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[I]:= (Random-Random)*100;
  Writeln( 'Max      : ', MaxValue( ExArray ):8:4 );
  Writeln( 'Min      : ', MinValue( ExArray ):8:4 );
  Writeln( 'Sum      : ', Sum( ExArray ):8:4 );
  Writeln( 'Sum (b) : ', Sum( @ExArray[1], 100 ):8:4 );
end.
```

sumofsquares

Declaration: `Function sumofsquares(const data : array of float) : float; Function sumofsquares(const data : PFloat; Const N : Integer) : float;`

Description: `Sumofsquares` returns the sum of the squares of the values in the data array.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: [sum \(282\)](#), [sumsandsquares \(284\)](#), [totalvariance \(285\)](#), [variance \(286\)](#)

Listing: mathex/ex45.pp

```
Program Example45;

{ Program to demonstrate the SumOfSquares function. }

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[I]:= (Random-Random)*100;
  Writeln( 'Max      : ', MaxValue( ExArray ):8:4 );
  Writeln( 'Min      : ', MinValue( ExArray ):8:4 );
  Writeln( 'Sum squares : ', SumOfSquares( ExArray ):8:4 );
```

```
  Writeln('Sum squares (b) : ', SumOfSquares(@ExArray[1],100):8:4);  
end.
```

sumsandsquares

Declaration: Procedure sumsandsquares(const data : array of float; var sum,sumofsquares : float); Procedure sumsandsquares(const data : PFloat; Const N : Integer; var sum,sumofsquares : float);

Description: sumsandsquares calculates the sum of the values and the sum of the squares of the values in the data array and returns the results in sum and sumofsquares.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: sum ([282](#)), sumofsquares ([283](#)), totalvariance ([285](#)), variance ([286](#))

Listing: mathex/ex46.pp

Program Example45;

{ Program to demonstrate the SumOfSquares function. }

Uses math;

Type

TExArray = **Array**[1..100] **of** Float;

Var

l : Integer;
ExArray : TExArray;
s,ss : float;

begin

```
  Randomize;  
  for l:=1 to 100 do  
    ExArray[l]:=(Random-Random)*100;  
  Writeln('Max           : ',MaxValue(ExArray):8:4);  
  Writeln('Min           : ',MinValue(ExArray):8:4);  
  SumsAndSquares(ExArray,S,SS);  
  Writeln('Sum           : ',S:8:4);  
  Writeln('Sum squares    : ',SS:8:4);  
  SumsAndSquares(@ExArray[1],100,S,SS);  
  Writeln('Sum (b)        : ',S:8:4);  
  Writeln('Sum squares (b) : ',SS:8:4);  
end.
```

tan

Declaration: Function tan(x : float) : float;

Description: Tan returns the tangent of x.

Errors: If x (normalized) is pi/2 or 3pi/2 then an overflow will occur.

See also: tanh ([285](#)), arcsin ([260](#)), sincos ([281](#)), arccos ([259](#))

Listing: mathex/ex47.pp

Program Example47;

{ Program to demonstrate the Tan function. }

Uses math;

Procedure DoTan(Angle : Float);

begin

Write (' Angle : ', RadToDeg(Angle):8:6);

Writeln (' Tangent : ', Tan(Angle):8:6);

end;

begin

 DoTan(0);

 DoTan(**Pi**);

 DoTan(**Pi**/3);

 DoTan(**Pi**/4);

 DoTan(**Pi**/6);

end.

tanh

Declaration: Function tanh(x : float) : float;

Description: Tanh returns the hyperbolic tangent of x.

Errors: None.

See also: arcsin ([260](#)), sincos ([281](#)), arccos ([259](#))

Listing: mathex/ex48.pp

Program Example48;

{ Program to demonstrate the Tanh function. }

Uses math;

begin

writeln (tanh(0));

writeln (tanh(1));

writeln (tanh(-1));

end.

totalvariance

Declaration: Function totalvariance(const data : array of float) : float; Function totalvariance(const data : PFloat; Const N : Integer) : float;

Description: TotalVariance returns the total variance of the values in the data array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: [variance \(286\)](#), [stddev \(282\)](#), [mean \(272\)](#)

Listing: mathex/ex49.pp

Program Example49;

{ Program to demonstrate the TotalVariance function. }

Uses math;

Type

TExArray = **Array**[1..100] **of** Float;

Var

I : Integer;

ExArray : TExArray;

TV : float;

begin

Randomize;

for I:=1 **to** 100 **do**

ExArray[I] := (Random-~~Random~~)*100;

TV:=TotalVariance(ExArray);

Writeln('Total variance : ',TV:8:4);

TV:=TotalVariance(@ExArray[1],100);

Writeln('Total Variance (b) : ',TV:8:4);

end.

variance

Declaration: Function variance(const data : array of float) : float; Function variance(const data : PFloat; Const N : Integer) : float;

Description: Variance returns the variance of the values in the data array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: [totalvariance \(285\)](#), [stddev \(282\)](#), [mean \(272\)](#)

Listing: mathex/ex50.pp

Program Example50;

{ Program to demonstrate the Variance function. }

Uses math;

Type

TExArray = **Array**[1..100] **of** Float;

Var

I : Integer;

ExArray : TExArray;

V : float;


```
begin
  Randomize;
  for i:=1 to 100 do
    ExArray[i]:=(Random-Random)*100;
  V:=Variance(ExArray);
  Writeln('Variance      : ',V:8:4);
  V:=Variance(@ExArray[1],100);
  Writeln('Variance (b) : ',V:8:4);
end.
```

Chapter 14

The MMX unit

This chapter describes the MMX unit. This unit allows you to use the MMX capabilities of the Free Pascal compiler. It was written by Florian Klämpfl for the i386 processor. It should work on all platforms that use the Intel processor.

14.1 Variables, Types and constants

The following types are defined in the MMX unit:

```
tmmxshortint = array[0..7] of shortint;  
tmmxbyte = array[0..7] of byte;  
tmmxword = array[0..3] of word;  
tmmxinteger = array[0..3] of integer;  
tmmxfixed = array[0..3] of fixed16;  
tmmxlongint = array[0..1] of longint;  
tmmxcardinal = array[0..1] of cardinal;  
{ for the AMD 3D }  
tmmxsingle = array[0..1] of single;
```

And the following pointers to the above types:

```
pmmxshortint = ^tmmxshortint;  
pmmxbyte = ^tmmxbyte;  
pmmxword = ^tmmxword;  
pmmxinteger = ^tmmxinteger;  
pmmxfixed = ^tmmxfixed;  
pmmxlongint = ^tmmxlongint;  
pmmxcardinal = ^tmmxcardinal;  
{ for the AMD 3D }  
pmmxsingle = ^tmmxsingle;
```

The following initialized constants allow you to determine if the computer has MMX extensions. They are set correctly in the unit's initialization code.

```
is_mmx_cpu : boolean = false;  
is_amd_3d_cpu : boolean = false;
```

14.2 Functions and Procedures

Emms

Declaration: Procedure Emms ;

Description: Emms sets all floating point registers to empty. This procedure must be called after you have used any MMX instructions, if you want to use floating point arithmetic. If you just want to move floating point data around, it isn't necessary to call this function, the compiler doesn't use the FPU registers when moving data. Only when doing calculations, you should use this function.

Errors: None.

See also: [Programmers guide](#)

Example:: Program MMXDemo;

```
uses mmx;
var
  d1 : double;
  a : array[0..10000] of double;
  i : longint;
begin
  d1:=1.0;
  {$mmx+}
  { floating point data is used, but we do _no_ arithmetic }
  for i:=0 to 10000 do
    a[i]:=d2; { this is done with 64 bit moves }
  {$mmx-}
  emms; { clear fpu }
  { now we can do floating point arithmetic again }
end.
```

Chapter 15

The MOUSE unit

The `Mouse` unit implements a platform independent mouse handling interface. It is implemented identically on all platforms supported by Free Pascal and can be enhanced with custom drivers, should this be needed.

15.1 Constants, Types and Variables

Constants

The following constants can be used when mouse drivers need to report errors:

```
const
  { We have an errorcode base of 1030 }
  errMouseBase           = 1030;
  errMouseInitError      = errMouseBase + 0;
  errMouseNotImplemented = errMouseBase + 1;
```

The following constants describe which action a mouse event describes

```
const
  MouseActionDown = $0001; { Mouse down event }
  MouseActionUp   = $0002; { Mouse up event }
  MouseActionMove = $0004; { Mouse move event }
```

The following constants describe the used buttons in a mouse event:

```
  MouseLeftButton  = $01; { Left mouse button }
  MouseRightButton = $02; { Right mouse button }
  MouseMiddleButton = $04; { Middle mouse button }
```

The mouse unit has a mechanism to buffer mouse events. The following constant defines the size of the event buffer:

```
MouseEventBufSize = 16;
```

Types

The `TMouseEvent` is the central type of the mouse unit, it is used to describe the mouse events:

```
PMouseEvent = ^TMouseEvent;  
TMouseEvent = packed record { 8 bytes }  
  buttons : word;  
  x,y      : word;  
  Action   : word;  
end;
```

The Buttons field describes which buttons were down when the event occurred. The x,y fields describe where the event occurred on the screen. The Action describes what action was going on when the event occurred. The Buttons and Action field can be examined using the above constants.

The following record is used to implement a mouse driver in the [SetMouseDriver \(296\)](#) function:

```
TMouseDriver = Record  
  UseDefaultQueue : Boolean;  
  InitDriver : Procedure;  
  DoneDriver : Procedure;  
  DetectMouse : Function : Byte;  
  ShowMouse : Procedure;  
  HideMouse : Procedure;  
  GetMouseX : Function : Word;  
  GetMouseY : Function : Word;  
  GetMouseButtons : Function : Word;  
  SetMouseXY : procedure (x,y:word);  
  GetMouseEvent : procedure (var MouseEvent:TMouseEvent);  
  PollMouseEvent : function (var MouseEvent: TMouseEvent):boolean;  
  PutMouseEvent : procedure (Const MouseEvent:TMouseEvent);  
end;
```

Its fields will be explained in the section on writing a custom driver.

Variables

The following variables are used to keep the current position and state of the mouse.

```
MouseIntFlag : Byte; { Mouse in int flag }  
MouseButtons : Byte; { Mouse button state }  
MouseWhereX,  
MouseWhereY : Word; { Mouse position }
```

15.2 Functions and procedures

DetectMouse

Declaration: `Function DetectMouse:byte;`

Description: `DetectMouse` detects whether a mouse is attached to the system or not. If there is no mouse, then zero is returned. If a mouse is attached, then the number of mouse buttons is returned.

This function should be called after the mouse driver was initialized.

Errors: None.

See also: [InitMouse \(304\)](#), [DoneMouse \(292\)](#),

Listing: mouseex/ex1.pp

Program Example1;*{ Program to demonstrate the DetectMouse function. }***Uses** mouse;**Var**

Buttons : Byte;

begin

InitMouse;

Buttons:=DetectMouse;

If Buttons=0 **then** **WriteLn**('No mouse present.')**else** **WriteLn**('Found mouse with ',Buttons,' buttons.');

DoneMouse;

end.

DoneMouse

Declaration: Procedure DoneMouse;

Description: DoneMouse De-initializes the mouse driver. It cleans up any memory allocated when the mouse was initialized, or removes possible mouse hooks from memory. The mouse functions will not work after DoneMouse was called. If DoneMouse is called a second time, it will exit at once. InitMouse should be called before DoneMouse can be called again.

Errors: None.

See also: DetectMouse ([291](#)), InitMouse ([304](#))

For an example, see most other mouse functions.

GetMouseButtons

Declaration: Function GetMouseButtons:word;

Description: GetMouseButtons returns the current button state of the mouse, i.e. it returns a or-ed combination of the following constants:

MouseLeftButtonWhen the left mouse button is held down.**MouseRightButton**When the right mouse button is held down.**MouseMiddleButton**When the middle mouse button is held down.

Errors: None.

See also: GetMouseEvent ([293](#)), GetMouseX ([293](#)), GetMouseY ([294](#))**Listing:** mouseex/ex2.pp

Program Example2;*{ Program to demonstrate the GetMouseButtons function. }*

Uses mouse;

begin

 InitMouse;

WriteLn('Press right mouse button to exit program');

While (GetMouseButtons<>MouseRightButton) **do** ;

 DoneMouse;

end.

GetMouseDriver

Declaration: Procedure GetMouseDriver(Var Driver : TMouseDriver);

Description: GetMouseDriver returns the currently set mouse driver. It can be used to retrieve the current mouse driver, and override certain callbacks.

A more detailed explanation about getting and setting mouse drivers can be found in section [15.3](#), page [297](#).

Errors: None.

See also: SetMouseDriver ([296](#))

For an example, see the section on writing a custom mouse driver, section [15.3](#), page [297](#)

GetMouseEvent

Declaration: Procedure GetMouseEvent(var MouseEvent : TMouseEvent);

Description: GetMouseEvent returns the next mouse event (a movement, button press or button release), and waits for one if none is available in the queue.

Some mouse drivers can implement a mouse event queue which can hold multiple events till they are fetched.; Others don't, and in that case, a one-event queue is implemented for use with PollMouseEvent ([295](#)).

Errors: None.

See also: GetMouseButtons ([292](#)), GetMouseX ([293](#)), GetMouseY ([294](#))

GetMouseX

Declaration: Function GetMouseX:word;

Description: GetMouseX returns the current X position of the mouse. X is measured in characters, starting at 0 for the left side of the screen.

Errors: None.

See also: GetMouseButtons ([292](#)), GetMouseEvent ([293](#)), GetMouseY ([294](#))

Listing: mouseex/ex4.pp

Program Example4;

{ Program to demonstrate the GetMouseX, GetMouseY functions. }

```
Uses mouse;

Var
  X,Y : Word;

begin
  InitMouse;
  Writeln('Move mouse cursor to square 10,10 to end');
  Repeat
    X:=GetMouseX;
    Y:=GetMouseY;
    Writeln('X,Y= (' ,X, ', ',Y, ') ');
  Until (X=9) and (Y=9);
  DoneMouse;
end.
```

GetMouseY

Declaration: `Function GetMouseY:word;`

Description: `GetMouseY` returns the current Y position of the mouse. Y is measured in characters, starting at 0 for the top of the screen.

Errors: None.

See also: `GetMouseButtons` ([292](#)), `GetMouseEvent` ([293](#)), `GetMouseX` ([293](#))

For an example, see `GetMouseX` ([293](#))

HideMouse

Declaration: `Procedure HideMouse;`

Description: `HideMouse` hides the mouse cursor. This may or may not be implemented on all systems, and depends on the driver.

Errors: None.

See also: `ShowMouse` ([311](#))

Listing: mouseex/ex5.pp

Program Example5;

{ Program to demonstrate the HideMouse function. }

Uses mouse;

Var
 Event : TMouseEvent;
 Visible : Boolean;

begin
 InitMouse;
 ShowMouse;
 Visible:=True;
 Writeln('Press left mouse button to hide/show, right button quits');


```
Repeat
  GetMouseEvent(Event);
With Event do
  If ( Buttons=MouseLeftbutton ) and
    ( Action=MouseDown ) then
    begin
    If Visible then
      HideMouse;
    else
      ShowMouse;
      Visible:=Not Visible;
    end;
  Until ( Event.Buttons=MouseRightButton ) and
    ( Event.Action=MouseDown );
  DoneMouse;
end.
```

InitMouse

Declaration: Procedure InitMouse;

Description: InitMouse initializes the mouse driver. This will allocate any data structures needed for the mouse to function. All mouse functions can be used after a call to InitMouse.

A call to InitMouse must always be followed by a call to DoneMouse (292) at program exit. Failing to do so may leave the mouse in an unusable state, or may result in memory leaks.

Errors: None.

See also: DoneMouse (292), DetectMouse (291)

For an example, see most other functions.

PollMouseEvent

Declaration: Function PollMouseEvent(var MouseEvent: TMouseEvent):boolean;

Description: PollMouseEvent checks whether a mouse event is available, and returns it in MouseEvent if one is found. The function result is True in that case. If no mouse event is pending, the function result is False, and the contents of MouseEvent is undefined.

Note that after a call to PollMouseEvent, the event should still be removed from the mouse event queue with a call to GetMouseEvent.

Errors: None.

See also: GetMouseEvent (293), PutMouseEvent (295)

PutMouseEvent

Declaration: Procedure PutMouseEvent(const MouseEvent: TMouseEvent);

Description: PutMouseEvent adds MouseEvent to the input queue. The next call to GetMouseEvent (293) or PollMouseEvent will then return MouseEvent.

Please note that depending on the implementation the mouse event queue can hold only one value.

Errors: None.

See also: GetMouseEvent (293), PollMouseEvent (295)

SetMouseDriver

Declaration: `Procedure SetMouseDriver(Const Driver : TMouseDriver);`

Description: `SetMouseDriver` sets the mouse driver to `Driver`. This function should be called before `InitMouse` (304) is called, or after `DoneMouse` is called. If it is called after the mouse has been initialized, it does nothing.

For more information on setting the mouse driver, section 15.3, page 297.

Errors:

See also: `InitMouse` (304), `DoneMouse` (292), `GetMouseDriver` (293)

For an example, see section 15.3, page 297

SetMouseXY

Declaration: `Procedure SetMouseXY(x,y:word);`

Description: `SetMouseXY` places the mouse cursor on `X`, `Y`. `X` and `Y` are zero based character coordinates: 0, 0 is the top-left corner of the screen, and the position is in character cells (i.e. not in pixels).

Errors: None.

See also: `GetMouseX` (293), `GetMouseY` (294)

Listing: mouseex/ex7.pp

Program Example7;

{ Program to demonstrate the SetMouseXY function. }

Uses mouse;

Var

Event : TMouseEvent;

begin

InitMouse;

Writeln('Click right mouse button to quit.');

SetMouseXY(40,12);

Repeat

If (GetMouseX>70) **then**

 SetMouseXY(10,GetMouseY);

If (GetMouseY>20) **then**

 SetMouseXY(GetMouseX,5);

 GetMouseEvent(Event);

Until (Event.Buttons=MouseRightButton) **and**
 (Event.Action=MouseActionDown);

 DoneMouse;

end.

ShowMouse

Declaration: `Procedure ShowMouse;`

Description: `ShowMouse` shows the mouse cursor if it was previously hidden. The capability to hide or show the mouse cursor depends on the driver.

Errors: None.

See also: [HideMouse \(304\)](#)

For an example, see [HideMouse \(304\)](#)

15.3 Writing a custom mouse driver

The `mouse` has support for adding a custom mouse driver. This can be used to add support for mice not supported by the standard Free Pascal driver, but also to enhance an existing driver for instance to log mouse events or to implement a record and playback function.

The following unit shows how a mouse driver can be enhanced by adding some logging capabilities to the driver.

Listing: `mouseex/logmouse.pp`

```
unit logmouse;  
  
interface  
  
Procedure StartMouseLogging;  
Procedure StopMouseLogging;  
Function IsMouseLogging : Boolean;  
Procedure SetMouseLogFileName (FileName : String);  
  
implementation  
  
uses sysutils, Mouse;  
  
var  
    NewMouseDriver,  
    OldMouseDriver : TMouseDriver;  
    Active, Logging : Boolean;  
    LogFileName : String;  
    MouseLog : Text;  
  
Function TimeStamp : String;  
  
begin  
    TimeStamp:=FormatDateTime('hh:nn:ss',Time());  
end;  
  
Procedure StartMouseLogging;  
  
begin  
    Logging:=True;  
    WriteLn(MouseLog,'Start logging mouse events at: ',TimeStamp);  
end;  
  
Procedure StopMouseLogging;  
  
begin  
    WriteLn(MouseLog,'Stop logging mouse events at: ',TimeStamp);  
    Logging:=False;  
end;
```

```
Function IsMouseLogging : Boolean;

begin
    IsMouseLogging:=Logging;
end;

Procedure LogGetMouseEvent(Var Event : TMouseEvent);

Var
    M : TMouseEvent;

begin
    OldMouseDriver.GetMouseEvent(M);
    If Logging then
        begin
            Write(MouseLog,TimeStamp,' : Mouse ');
            With M do
                begin
                    Case Action of
                        MouseActionDown : Write(MouseLog,'down');
                        MouseActionUp   : Write(MouseLog,'up');
                        MouseActionMove  : Write(MouseLog,'move');
                    end;
                Write(MouseLog,' event at ',X,', ',Y);
                If (Buttons<>0) then
                    begin
                        Write(MouseLog,' for buttons: ');
                        If (Buttons and MouseLeftbutton)<>0 then
                            Write(MouseLog,' Left ');
                        If (Buttons and MouseRightbutton)<>0 then
                            Write(MouseLog,' Right ');
                        If (Buttons and MouseMiddlebutton)<>0 then
                            Write(MouseLog,' Middle ');
                        end;
                    WriteLn(MouseLog);
                end;
            end;
        end;

Procedure LogInitMouse;

begin
    OldMouseDriver.InitDriver();
    Assign(MouseLog,logFileName);
    Rewrite(MouseLog);
    Active:=True;
    StartMouseLogging;
end;

Procedure LogDoneMouse;

begin
    StopMouseLogging;
    Close(MouseLog);
    Active:=False;
    OldMouseDriver.DoneDriver();
end;
```

Procedure SetMouseLogFileName (FileName : **String**);

begin

If Not Active **then**

 LogFileName:=FileName;

end;

Initialization

 GetMouseDriver (OldMouseDriver);

 NewMouseDriver:=OldMouseDriver;

 NewMouseDriver.GetMouseEvent:=@LogGetMouseEvent;

 NewMouseDriver.InitDriver:=@LogInitMouse;

 NewMouseDriver.DoneDriver:=@LogDoneMouse;

 LogFileName:='Mouse.log';

 Logging:=False;

 SetMouseDriver (NewMouseDriver);

end.

Chapter 16

The MsMouse unit

The msmouse unit provides basic mouse handling under DOS (Go32v1 and Go32v2) Some general remarks about the msmouse unit:

- For maximum portability, it is advisable to use the **mouse** unit; that unit is portable across platforms, and offers a similar interface. Under no circumstances should the two units be used together.
- The mouse driver does not know when the text screen scrolls. This results in unerased mouse cursors on the screen when the screen scrolls while the mouse cursor is visible. The solution is to hide the mouse cursor (using `HideMouse`) when you write something to the screen and to show it again afterwards (using `ShowMouse`).
- All Functions/Procedures that return and/or accept coordinates of the mouse cursor, always do so in pixels and zero based (so the upper left corner of the screen is (0,0)). To get the (column, row) in standard text mode, divide both x and y by 8 (and add 1 if you want to have it 1 based).
- The real resolution of graphic modes and the one the mouse driver uses can differ. For example, mode 13h (320*200 pixels) is handled by the mouse driver as 640*200, so you will have to multiply the X coordinates you give to the driver and divide the ones you get from it by 2 in that mode.
- By default the msmouse unit is compiled with the conditional define `MouseCheck`. This causes every procedure/function of the unit to check the `MouseFound` variable prior to doing anything. Of course this is not necessary, so if you are sure you are not calling any mouse unit procedures when no mouse is found, you can recompile the mouse unit without this conditional define.
- You will notice that several procedures/functions have longint sized parameters while only the lower 16 bits are used. This is because FPC is a 32 bit compiler and consequently 32 bit parameters result in faster code.

16.1 Constants, types and variables

The following constants are defined (to be used in e.g. the `GetLastButtonPress` (301) call).

```
LButton = 1; {left button}
RButton = 2; {right button}
MButton = 4; {middle button}
```

The following variable exist:

```
MouseFound: Boolean;
```

it is set to True or False in the unit's initialization code.

16.2 Functions and procedures

GetLastButtonPress

Declaration: `Function GetLastButtonPress (Button: Longint; Var x,y:Longint) : Longint;`

Description: `GetLastButtonPress` Stores the position where Button was last pressed in x and y and returns the number of times this button has been pressed since the last call to this function with Button as parameter. For Button you can use the `LButton`, `RButton` and `MButton` constants for resp. the left, right and middle button. With certain mouse drivers, checking the middle button when using a two-button mouse to gives and clears the stats of the right button.

Errors: None.

See also: `GetLastButtonRelease` ([302](#))

Listing: `mmouseex/mouse5.pp`

{example for GetLastButtonPress and GetLastButtonRelease}

Uses `MsMouse`, `Crt`;

Var `x`, `y`, `times`: `Longint`;
 `c`: `Char`;

Begin

If `MouseFound` **Then**

Begin

`ClrScr`;

`ShowMouse`;

`WriteLn`('Move the mouse and click the buttons (press escape to quit).');

`WriteLn`('Press the L-key to see the stats for the left button.');

`WriteLn`('Press the R-key to see the stats for the right button.');

`WriteLn`('Press the M-key to see the stats for the middle button.');

`GotoXY`(1,19);

`Write`('Since the last call to `GetLastButtonPress` with this button as parameter, the');

`GotoXY`(1,22);

`Write`('Since the last call to `GetLastButtonRelease` with this button as parameter, the');

Repeat

If `Keypressed` **Then**

Begin

`c` := `UpCase`(`Readkey`);

Case `c` **Of**

 'L':

Begin

`GotoXY`(1, 20);

`ClrEol`;

`times` := `GetLastButtonPress`(`LButton`, `x`, `y`);

`Write`('left button has been pressed ',`times`,
 ' times, the last time at (' ,`x`, ',' ,`y`, ')');

`times` := `GetLastButtonRelease`(`LButton`, `x`, `y`);

`GotoXY`(1,23);

`ClrEol`;

```
        Write('left button has been released ',times,
              ' times, the last time at (' ,x, ',' ,y, ')')
    End;
'R':
    Begin
        GotoXY(1, 20);
        ClrEol;
        times := GetLastButtonPress(RButton, x, y);
        WriteIn('right button has been pressed ',times,
              ' times, the last time at (' ,x, ',' ,y, ')');
        times := GetLastButtonRelease(RButton, x, y);
        GotoXY(1,23);
        ClrEol;
        Write('right button has been released ',times,
              ' times, the last time at (' ,x, ',' ,y, ')')
    End;
'M':
    Begin
        GotoXY(1, 20);
        ClrEol;
        times := GetLastButtonPress(MButton, x, y);
        WriteIn('middle button has been pressed ',times,
              ' times, the last time at (' ,x, ',' ,y, ')');
        times := GetLastButtonRelease(MButton, x, y);
        GotoXY(1,23);
        ClrEol;
        Write('middle button has been released ',times,
              ' times, the last time at (' ,x, ',' ,y, ')')
    End
End
End;
Until (c = #27); {escape}
While KeyPressed do ReadKey;
GotoXY(1,24);
HideMouse
End
End.
```

GetLastButtonRelease

Declaration: Function GetLastButtonRelease (Button: Longint; Var x,y:Longint) : Longint;

Description: GetLastButtonRelease stores the position where Button was last released in x and y and returns the number of times this button has been released since the last call to this function with Button as parameter. For button you can use the LButton, RButton and MButton constants for resp. the left, right and middle button. With certain mouse drivers, checking the middle button when using a two-button mouse to gives and clears the stats of the right button.

Errors: None.

See also: [GetLastButtonPress \(301\)](#)

For an example, see [GetLastButtonPress \(301\)](#).

GetMouseState

Declaration: Procedure GetMouseState (Var x, y, buttons: Longint);

Description: GetMouseState Returns information on the current mouse position and which buttons are currently pressed. x and y return the mouse cursor coordinates in pixels. Buttons is a bitmask. Check the example program to see how you can get the necessary information from it.

Errors: None.

See also: LPressed (305), MPressed (305), RPressed (305), SetMousePos (307)

Listing: mmouseex/mouse3.pp

{example for GetMouseState, IsLPressed, IsRPressed and IsMPressed}

Uses MsMouse, Crt;

Var X, Y, State: Longint;

Begin

If MouseFound **Then**

Begin

 ClrScr;

 ShowMouse;

 GotoXY(5,24);

 Write(' Left button:');

 GotoXY(30,24);

 Write(' Right button:');

 GotoXY(55,24);

 Write(' Middle button:');

While KeyPressed **do** Readkey; *{clear keyboard buffer}*

Repeat

 GetMouseState(x, y, State);

 GotoXY(20, 22);

 Write('X: ',x:5, ' (column: ',(x div 8):2,') Y: ',y:5, ' (row: ',(y div 8):2,')');

 GotoXY(18, 24); *{left button}*

If (State and LButton) = LButton **Then**

{or: " If LPressed Then". If you use this function, no call to GetMouseState is necessary}

 Write('Down')

Else

 Write('Up ');

 GotoXY(44, 24); *{right button}*

If (State and RButton) = RButton **Then**

{or: " If RPressed Then"}

 Write('Down')

Else

 Write('Up ');

 GotoXY(70, 24); *{middle button}*

If (State and MButton) = MButton **Then**

{or: " If MPressed Then"}

 Write('Down')

Else

 Write('Up ');

Until KeyPressed;

 HideMouse;

While KeyPressed **Do** Readkey

End

End.

HideMouse

Declaration: `Procedure HideMouse ;`

Description: `HideMouse` makes the mouse cursor invisible. Multiple calls to `HideMouse` will require just as many calls to `ShowMouse` to make the mouse cursor visible again.

Errors: None.

See also: `ShowMouse` ([311](#)), `SetMouseHideWindow` ([306](#))

For an example, see `ShowMouse` ([311](#)).

InitMouse

Declaration: `Procedure InitMouse ;`

Description: `InitMouse` initializes the mouse driver sets the variable `MouseFound` depending on whether or not a mouse is found. This is automatically called at the start of your program. You should never have to call it, unless you want to reset everything to its default values.

Errors: None.

See also: `MouseFound` variable.

Listing: `mmouseex/mouse1.pp`

Program `Mouse1;`

{example for InitMouse and MouseFound}

Uses `MsMouse;`

Begin

If `MouseFound` **Then**

Begin

{go into graphics mode 13h}

Asm

`movl $0x013, %eax`

`pushl %ebp`

`int $0x010`

`popl %ebp`

End;

`InitMouse;`

`ShowMouse; {otherwise it stays invisible}`

`WriteLn('Mouse Found! (press enter to quit)');`

ReadLn;

{back to text mode}

Asm

`movl $3, %eax`

`pushl %ebp`

`int $0x010`

`popl %ebp`

End

End

End.

LPressed

Declaration: `Function LPressed : Boolean;`

Description: `LPressed` returns `True` if the left mouse button is pressed. This is simply a wrapper for the `GetMouseState` procedure.

Errors: None.

See also: `GetMouseState` (303), `MPressed` (305), `RPressed` (305)

For an example, see `GetMouseState` (303).

MPressed

Declaration: `Function MPressed : Boolean;`

Description: `MPressed` returns `True` if the middle mouse button is pressed. This is simply a wrapper for the `GetMouseState` procedure.

Errors: None.

See also: `GetMouseState` (303), `LPressed` (305), `RPressed` (305)

For an example, see `GetMouseState` (303).

RPressed

Declaration: `Function RPressed : Boolean;`

Description: `RPressed` returns `True` if the right mouse button is pressed. This is simply a wrapper for the `GetMouseState` procedure.

Errors: None.

See also: `GetMouseState` (303), `LPressed` (305), `MPressed` (305)

For an example, see `GetMouseState` (303).

SetMouseAscii

Declaration: `Procedure SetMouseAscii (Ascii: Byte);`

Description: `SetMouseAscii` sets the `Ascii` value of the character that depicts the mouse cursor in text mode. The difference between this one and `SetMouseShape` (308), is that the foreground and background colors stay the same and that the `Ascii` code you enter is the character that you will get on screen; there's no XOR'ing.

Errors: None

See also: `SetMouseShape` (308)

Listing: `mmouseex/mouse8.pp`

```
{example for SetMouseAscii}

{warning: no error checking is performed on the input}

Uses MsMouse, Crt;

Var ascii: Byte;
    x,y: Longint;

Begin
  If MouseFound Then
    Begin
      ClrScr;
      WriteLn('Press any mouse button to quit after you've entered an Ascii value. ');
      WriteLn;
      WriteLn('ASCII value of mouse cursor: ');
      ShowMouse;
      Repeat
        GotoXY(30,3);
        ClrEol;
        ReadLn(ascii);
        SetMouseAscii(ascii)
      Until ( GetLastButtonPress(LButton,x,y) <> 0) Or
            ( GetLastButtonPress(RButton,x,y) <> 0) Or
            ( GetLastButtonPress(MButton,x,y) <> 0);
      HideMouse
    End;
End.
```

SetMouseHideWindow

Declaration: Procedure SetMouseHideWindow (xmin,ymin,xmax,ymax: Longint);

Description: SetMouseHideWindow defines a rectangle on screen with top-left corner at (xmin,ymin) and botto-right corner at (xmax,ymax),which causes the mouse cursor to be turned off when it is moved into it. When the mouse is moved into the specified region, it is turned off until you call ShowMouse again. However, once you've called ShowMouse (311), you'll have to call SetMouseHideWindow again to redefine the hide window... This may be annoying, but it's the way it's implemented in the mouse driver. While xmin, ymin, xmax and ymax are Longint parameters, only the lower 16 bits are used.

Warning: it seems Win98 SE doesn't (properly) support this function, maybe this already the case with earlier versions too!

Errors: None.

See also: ShowMouse (311), HideMouse (304)

Listing: mmouseex/mouse9.pp

```
{example for SetMouseHideWindow}

{warning: when the mouse is moved into the specified region, it is turned off
until you call ShowMouse again. However, when you've called ShowMouse,
you'll have to call SetMouseHideWindow again to redefine the hide window...
It's not our fault, that's the way it's implemented in the mouse driver.
```

Below you can find an example of how to define a "permanent" hide region with the cursor showing up again when you move it out of the region

Note: the mouse functions are zero-based, GotoXY is 1-based}

```
Uses MsMouse, Crt;  
  
Var x, y, buttons: Longint;  
      MouseOn: Boolean;  
  
Begin  
  If MouseFound Then  
    Begin  
      ClrScr;  
      For y := 1 to 25 Do  
        Begin  
          GotoXY(20,y);  
          Write('|');  
          GotoXY(60,y);  
          Write('|');  
        End;  
      MouseOn := true;  
      GotoXY(30, 24);  
      WriteLn('Press any key to quit');  
      ShowMouse;  
      SetMousePos(1,1);  
      While KeyPressed Do Readkey;  
      Repeat  
        GetMouseState(x,y,buttons);  
        If Not(MouseOn) And  
          ((x <= 19*8) or (x >= 59*8)) Then  
          Begin  
            ShowMouse;  
            MouseOn := true  
          End;  
        If MouseOn And (x > 19*8) And (x<59*8) Then  
          Begin  
            SetMouseHideWindow(20*8,0,60*8,25*8);  
            MouseOn := false  
          End;  
      Until KeyPressed;  
      While KeyPressed Do Readkey;  
      HideMouse  
    End  
End.
```

SetMousePos

Declaration: Procedure SetMousePos (x,y:Longint);

Description: SetMosusePos sets the position of the mouse cursor on the screen. x is the horizontal position in pixels, y the vertical position in pixels. The upper-left hand corner of the screen is the origin. While x and y are longints, only the lower 16 bits are used.

Errors: None.

See also: GetMouseState ([303](#))

Listing: mmouseex/mouse4.pp

{example for SetMousePos}

Uses MsMouse, Crt;

Begin

If MouseFound **Then**

Begin

 ShowMouse;

While KeyPressed **do** ReadKey;

Repeat

 SetMousePos(Random(80*8), Random(25*8));

 delay(100);

Until Keypressed;

 HideMouse;

While KeyPressed **do** ReadKey;

End;

End.

SetMouseShape

Declaration: Procedure SetMouseShape (ForeColor,BackColor,Ascii: Byte);

Description: SetMouseShape defines how the mouse cursor looks in textmode The character and its attributes that are on the mouse cursor's position on screen are XOR'ed with resp. ForeColor, BackColor and Ascii. Set them all to 0 for a "transparent" cursor.

Errors: None.

See also: SetMouseAscii ([305](#))

Listing: mmouseex/mouse7.pp

{example for SetMouseShape}

{warning: no error checking is performed on the input}

{the Ascii value you enter is XOR'ed with the Ascii value of the character on the screen over which you move the cursor. To get a "transparent" cursor, use the Ascii value 0}

Uses MsMouse, Crt;

Var ascii, fc, bc: Byte;
 x,y: Longint;

Begin

If MouseFound **Then**

Begin

 ClrScr;

 Writeln('Press any mouse button to quit after you''ve entered a sequence of numbers.');

 Writeln;

 Writeln('ASCII value of mouse cursor:');

 Writeln('Foreground color:');

 Writeln('Background color:');

 ShowMouse;

Repeat

```
    GotoXY(30,3);
    ClrEol;
    Readln(ascii);
    GotoXY(18,4);
    ClrEol;
    Readln(fc);
    GotoXY(19,5);
    ClrEol;
    Readln(bc);
    SetMouseShape(fc, bc, ascii)
  Until ( GetLastButtonPress(LButton,x,y) <> 0) Or
        ( GetLastButtonPress(RButton,x,y) <> 0) Or
        ( GetLastButtonPress(MButton,x,y) <> 0);
  HideMouse
End;
End.
```

SetMouseSpeed

Declaration: Procedure SetMouseSpeed (Horizontal, Vertical: Longint);

Description: SetMouseSpeed sets the mouse speed in mickeys per 8 pixels. A mickey is the smallest measurement unit handled by a mouse. With this procedure you can set how many mickeys the mouse should move to move the cursor 8 pixels horizontally or vertically. The default values are 8 for horizontal and 16 for vertical movement. While this procedure accepts longint parameters, only the low 16 bits are actually used.

Errors: None.

See also:

Listing: mmouseex/mouse10.pp

Uses MsMouse, Crt;

Var hor, vert: Longint;
x, y: Longint;

Begin

If MouseFound **Then**

Begin

ClrScr;

WriteLn('Click any button to quit after you''ve entered a sequence of numbers.');

WriteLn;

WriteLn('Horizontal mickey''s per pixel:');

WriteLn('Vertical mickey''s per pixel:');

ShowMouse;

Repeat

GotoXY(32,3);

ClrEol;

Readln(hor);

GotoXY(30,4);

ClrEol;

Readln(vert);

SetMouseSpeed(hor, vert);

Until (GetLastButtonPress(LButton,x,y) <> 0) **Or**

(GetLastButtonPress(RButton,x,y) <> 0) **Or**

(GetLastButtonPress(MButton,x,y) <> 0);

End
End.

SetMouseWindow

Declaration: `Procedure SetMouseWindow (xmin,ymin,xmax,ymax: Longint);`

Description: SetMouseWindow defines a rectangle on screen with top-left corner at (xmin,ymin) and bottom-right corner at (xmax,ymax), out of which the mouse cursor can't move. This procedure is simply a wrapper for the SetMouseXRange (310) and SetMouseYRange (311) procedures. While xmin, ymin, xmax and ymax are Longint parameters, only the lower 16 bits are used.

Errors: None.

See also: SetMouseXRange (310), SetMouseYRange (311)

For an example, see SetMouseXRange (310).

SetMouseXRange

Declaration: `Procedure SetMouseXRange (Min, Max: Longint);`

Description: SetMouseXRange sets the minimum (Min) and maximum (Max) horizontal coordinates in between which the mouse cursor can move. While Min and Max are Longint parameters, only the lower 16 bits are used.

Errors: None.

See also: SetMouseYRange (311), SetMouseWindow (310)

Listing: mmouseex/mouse6.pp

{example for SetMouseXRange, SetMouseYRange and SetMouseWindow}

Uses MsMouse, Crt;

Begin

If MouseFound **Then**

Begin

SetMouseXRange(20*8,50*8); *{character width and height = 8 pixels}*
SetMouseYRange(10*8,15*8);

{the two lines of code have exactly the same effect as
*SetMouseWindow(20*8,10*8,50*8,15*8)}*

WriteIn('Press any key to quit.');

ShowMouse;

While KeyPressed **Do** ReadKey;

Readkey;

While KeyPressed **Do** ReadKey;

HideMouse

End

End.

SetMouseYRange

Declaration: `Procedure SetMouseYRange (Min, Max: Longint);`

Description: `SetMouseYRange` sets the minimum (Min) and maximum (Max) vertical coordinates in between which the mouse cursor can move. While Min and Max are Longint parameters, only the lower 16 bits are used.

Errors: None.

See also: `SetMouseXRange` (310), `SetMouseWindow` (310)

For an example, see `SetMouseXRange` (310).

ShowMouse

Declaration: `Procedure ShowMouse ;`

Description: `ShowMouse` makes the mouse cursor visible. At the start of your program, the mouse cursor is invisible.

Errors: None.

See also: `HideMouse` (304), `SetMouseHideWindow` (306)

Listing: `mmouseex/mouse2.pp`

{example for ShowMouse and HideMouse}

Uses MsMouse;

Begin

ClrScr;

If MouseFound **Then**

Begin

WriteLn('Now you can see the mouse... (press enter to continue)');

 ShowMouse;

ReadLn;

 HideMouse;

WriteLn('And now you can''t... (press enter to quit)');

ReadLn

End

End.

Chapter 17

The Objects unit.

This chapter documents the `objects` unit. The unit was implemented by many people, and was mainly taken from the FreeVision sources. It has been ported to all supported platforms.

The methods and fields that are in a `Private` part of an object declaration have been left out of this documentation.

17.1 Constants

The following constants are error codes, returned by the various stream objects.

```
CONST
    stOk          = 0; { No stream error }
    stError       = -1; { Access error }
    stInitError   = -2; { Initialize error }
    stReadError   = -3; { Stream read error }
    stWriteError  = -4; { Stream write error }
    stGetError    = -5; { Get object error }
    stPutError    = -6; { Put object error }
    stSeekError   = -7; { Seek error in stream }
    stOpenError   = -8; { Error opening stream }
```

These constants can be passed to constructors of file streams:

```
CONST
    stCreate      = $3C00; { Create new file }
    stOpenRead    = $3D00; { Read access only }
    stOpenWrite   = $3D01; { Write access only }
    stOpen        = $3D02; { Read/write access }
```

The following constants are error codes, returned by the collection list objects:

```
CONST
    coIndexError = -1; { Index out of range }
    coOverflow   = -2; { Overflow }
```

Maximum data sizes (used in determining how many data can be used).

```
CONST
    MaxBytes = 128*1024*1024;           { Maximum data size }
    MaxWords = MaxBytes DIV SizeOf(Word); { Max word data size }
    MaxPtrs = MaxBytes DIV SizeOf(Pointer); { Max ptr data size }
    MaxCollectionSize = MaxBytes DIV SizeOf(Pointer); { Max collection size }
```

17.2 Types

The following auxiliary types are defined:

```
TYPE
    { Character set }
    TCharSet = SET Of Char;
    PCharSet = ^TCharSet;

    { Byte array }
    TByteArray = ARRAY [0..MaxBytes-1] Of Byte;
    PByteArray = ^TByteArray;

    { Word array }
    TWordArray = ARRAY [0..MaxWords-1] Of Word;
    PWordArray = ^TWordArray;

    { Pointer array }
    TPointerArray = Array [0..MaxPtrs-1] Of Pointer;
    PPointerArray = ^TPointerArray;

    { String pointer }
    PString = ^String;

    { Filename array }
    AsciiZ = Array [0..255] Of Char;

    Sw_Word = Cardinal;
    Sw_Integer = LongInt;
```

The following records are used internally for easy type conversion:

```
TYPE
    { Word to bytes }
    WordRec = packed RECORD
        Lo, Hi: Byte;
    END;

    { LongInt to words }
    LongRec = packed RECORD
        Lo, Hi: Word;
    END;

    { Pointer to words }
    PtrRec = packed RECORD
        Ofs, Seg: Word;
    END;
```

The following record is used when streaming objects:

```
TYPE
  PStreamRec = ^TStreamRec;
  TStreamRec = Packed RECORD
    ObjType: Sw_Word;
    VmtLink: pointer;
    Load : Pointer;
    Store: Pointer;
    Next : PStreamRec;
  END;
```

The TPoint basic object is used in the TRect object (see section [17.4](#), page [317](#)):

```
TYPE
  PPoint = ^TPoint;
  TPoint = OBJECT
    X, Y: Sw_Integer;
  END;
```

17.3 Procedures and Functions

NewStr

Declaration: Function NewStr (Const S: String): PString;

Description: NewStr makes a copy of the string S on the heap, and returns a pointer to this copy.

The allocated memory is not based on the declared size of the string passed to NewStr, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr ([315](#))

```
Program ex40;

{ Program to demonstrate the NewStr function }

Uses Objects;

Var S : String;
    P : PString;

begin
  S:='Some really cute string';
  Writeln ( 'Memavail : ',Memavail);
  P:=NewStr(S);
  If P^<>S then
    Writeln ( 'Oh-oh... Something is wrong !!');
  Writeln ( 'Allocated string. Memavail : ',Memavail);
  DisposeStr(P);
  Writeln ( 'Deallocated string. Memavail : ',Memavail);
end.
```

DisposeStr

Declaration: `Procedure DisposeStr (P: PString);`

Description: `DisposeStr` removes a dynamically allocated string from the heap.

Errors: None.

See also: `NewStr` ([314](#))

For an example, see `NewStr` ([314](#)).

Abstract

Declaration: `Procedure Abstract;`

Description: When implementing abstract methods, do not declare them as `abstract`. Instead, define them simply as `virtual`. In the implementation of such abstract methods, call the `Abstract` procedure. This allows explicit control of what happens when an abstract method is called.

The current implementation of `Abstract` terminates the program with a run-time error 211.

Errors: None.

See also: Most abstract types.

RegisterObjects

Declaration: `Procedure RegisterObjects;`

Description: `RegisterObjects` registers the following objects for streaming:

1. `TCollection`, see section [17.10](#), page [341](#).
2. `TStringCollection`, see section [17.12](#), page [360](#).
3. `TStrCollection`, see section [17.13](#), page [361](#).

Errors: None.

See also: `RegisterType` ([315](#))

RegisterType

Declaration: `Procedure RegisterType (Var S: TStreamRec);`

Description: `RegisterType` registers a new type for streaming. An object cannot be streamed unless it has been registered first. The stream record `S` needs to have the following fields set:

ObjType: `Sw_Word` This should be a unique identifier. Each possible type should have its own identifier.

VmtLink: **pointer** This should contain a pointer to the VMT (Virtual Method Table) of the object you try to register. You can get it with the following expression:

`VmtLink: ofs(KindOf(MyType)^);`

Load : Pointer is a pointer to a method that initializes an instance of that object, and reads the initial values from a stream. This method should accept as its sole argument a `PStream` type variable.

Store: **Pointer** is a pointer to a method that stores an instance of the object to a stream. This method should accept as its sole argument a **PStream** type variable.

Errors: In case of error (if a object with the same **ObjType**) is already registered), run-time error 212 occurs.

Unit MyObject;

Interface

Uses Objects;

Type

```
PMYObject = ^TMyObject;
TMyObject = Object(TObject)
  Field : Longint;
  Constructor Init;
  Constructor Load (Var Stream : TStream);
  Destructor Done;
  Procedure Store (Var Stream : TStream);
  Function GetField : Longint;
  Procedure SetField (Value : Longint);
end;
```

Implementation

Constructor TMyobject.Init;

```
begin
  Inherited Init;
  Field := -1;
end;
```

Constructor TMyobject.Load (Var Stream : TStream);

```
begin
  Stream.Read(Field, Sizeof(Field));
end;
```

Destructor TMyObject.Done;

```
begin
end;
```

Function TMyObject.GetField : Longint;

```
begin
  GetField := Field;
end;
```

Procedure TMyObject.SetField (Value : Longint);

```
begin
  Field := Value;
end;
```

Procedure TMyObject.Store (Var Stream : TStream);

```
begin
  Stream.Write(Field, SizeOf(Field));
end;

Const MyObjectRec : TStreamRec = (
  Objtype : 666;
  vmtlink : Ofs(TypeOf(TMyObject)^);
  Load : @TMyObject.Load;
  Store : @TMyObject.Store;
);

begin
  RegisterObjects;
  RegisterType(MyObjectRec);
end.
```

LongMul

Declaration: Function LongMul (X, Y: Integer): LongInt;

Description: LongMul multiplies X with Y. The result is of type Longint. This avoids possible overflow errors you would normally get when multiplying X and Y that are too big.

Errors: None.

See also: LongDiv ([317](#))

LongDiv

Declaration: Function LongDiv (X: Longint; Y: Integer): Integer;

Description: LongDiv divides X by Y. The result is of type Integer instead of type Longint, as you would get normally.

Errors: If Y is zero, a run-time error will be generated.

See also: LongMul ([317](#))

17.4 TRect

The TRect object is declared as follows:

```
TRect = OBJECT
  A, B: TPoint;
  FUNCTION Empty: Boolean;
  FUNCTION Equals (R: TRect): Boolean;
  FUNCTION Contains (P: TPoint): Boolean;
  PROCEDURE Copy (R: TRect);
  PROCEDURE Union (R: TRect);
  PROCEDURE Intersect (R: TRect);
  PROCEDURE Move (ADX, ADY: Sw_Integer);
  PROCEDURE Grow (ADX, ADY: Sw_Integer);
  PROCEDURE Assign (XA, YA, XB, YB: Sw_Integer);
END;
```

TRect.Empty

Declaration: `Function TRect.Empty: Boolean;`

Description: `Empty` returns `True` if the rectangle defined by the corner points A, B has zero or negative surface.

Errors: None.

See also: `TRect.Equals` ([318](#)), `TRect.Contains` ([319](#))

```
Program ex1;

{ Program to demonstrate TRect.Empty }

Uses objects;

Var ARect,BRect : TRect;
    P : TPoint;

begin
  With ARect.A do
    begin
      X:=10;
      Y:=10;
    end;
  With ARect.B do
    begin
      X:=20;
      Y:=20;
    end;
  { Offset B by (5,5) }
  With BRect.A do
    begin
      X:=15;
      Y:=15;
    end;
  With BRect.B do
    begin
      X:=25;
      Y:=25;
    end;
  { Point }
  With P do
    begin
      X:=15;
      Y:=15;
    end;
  Writeln ( 'A empty : ',ARect.Empty);
  Writeln ( 'B empty : ',BRect.Empty);
  Writeln ( 'A Equals B : ',ARect.Equals(BRect));
  Writeln ( 'A Contains (15,15) : ',ARect.Contains(P));
end.
```

TRect.Equals

Declaration: `Function TRect.Equals (R: TRect): Boolean;`

Description: `Equals` returns `True` if the rectangle has the same corner points A,B as the rectangle R, and `False` otherwise.

Errors: None.

See also: `Empty` (318), `Contains` (319)

For an example, see `TRect.Empty` (318)

TRect.Contains

Declaration: `Function TRect.Contains (P: TPoint): Boolean;`

Description: `Contains` returns `True` if the point P is contained in the rectangle (including borders), `False` otherwise.

Errors: None.

See also: `Intersect` (320), `Equals` (318)

TRect.Copy

Declaration: `Procedure TRect.Copy (R: TRect);`

Description: Assigns the rectangle R to the object. After the call to `Copy`, the rectangle R has been copied to the object that invoked `Copy`.

Errors: None.

See also: `Assign` (322)

```
Program ex2;

{ Program to demonstrate TRect.Copy }

Uses objects;

Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  CRect.Copy(ARect);
  If ARect.Equals(CRect) Then
    WriteLn ( 'ARect equals CRect' )
  Else
    WriteLn ( 'ARect does not equal CRect !' );
end.
```

TRect.Union

Declaration: `Procedure TRect.Union (R: TRect);`

Description: `Union` enlarges the current rectangle so that it becomes the union of the current rectangle with the rectangle R.

Errors: None.

See also: [Intersect \(320\)](#)

```
Program ex3;

{ Program to demonstrate TRect.Union }

Uses objects;

Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  { CRect is union of ARect and BRect }
  CRect.Assign(10,10,25,25);
  { Calculate it explicitly }
  ARect.Union(BRect);
  If ARect.Equals(CRect) Then
    WriteLn ( 'ARect equals CRect' )
  Else
    WriteLn ( 'ARect does not equal CRect !' );
end.
```

TRect.Intersect

Declaration: `Procedure TRect.Intersect (R: TRect);`

Description: `Intersect` makes the intersection of the current rectangle with R. If the intersection is empty, then the rectangle is set to the empty rectangle at coordinate (0,0).

Errors: None.

See also: [Union \(319\)](#)

```
Program ex4;

{ Program to demonstrate TRect.Intersect }

Uses objects;

Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  { CRect is intersection of ARect and BRect }
  CRect.Assign(15,15,20,20);
  { Calculate it explicitly }
  ARect.Intersect(BRect);
  If ARect.Equals(CRect) Then
    WriteLn ( 'ARect equals CRect' )
  Else
    WriteLn ( 'ARect does not equal CRect !' );
  BRect.Assign(25,25,30,30);
  ARect.Intersect(BRect);
  If ARect.Empty Then
```

```
    Writeln ( 'ARect is empty' );  
end.
```

TRect.Move

Declaration: `Procedure TRect.Move (ADX, ADY: Sw_Integer);`

Description: `Move` moves the current rectangle along a vector with components (ADX, ADY). It adds ADX to the X-coordinate of both corner points, and ADY to both end points.

Errors: None.

See also: `Grow` ([321](#))

```
Program ex5;  
  
    { Program to demonstrate TRect.Move }  
  
Uses objects;  
  
  
Var ARect, BRect : TRect;  
  
begin  
    ARect.Assign(10,10,20,20);  
    ARect.Move(5,5);  
    // Brect should be where new ARect is .  
    BRect.Assign(15,15,25,25);  
    If ARect.Equals(BRect) Then  
        Writeln ( 'ARect equals BRect' )  
    Else  
        Writeln ( 'ARect does not equal BRect !' );  
end.
```

TRect.Grow

Declaration: `Procedure TRect.Grow (ADX, ADY: Sw_Integer);`

Description: `Grow` expands the rectangle with an amount ADX in the X direction (both on the left and right side of the rectangle, thus adding a length 2*ADX to the width of the rectangle), and an amount ADY in the Y direction (both on the top and the bottom side of the rectangle, adding a length 2*ADY to the height of the rectangle).

ADX and ADY can be negative. If the resulting rectangle is empty, it is set to the empty rectangle at (0,0).

Errors: None.

See also: `Move` ([321](#)).

```
Program ex6;  
  
    { Program to demonstrate TRect.Grow }  
  
Uses objects;
```

```
Var ARect,BRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  ARect.Grow(5,5);
  // Brect should be where new ARect is.
  BRect.Assign(5,5,25,25);
  If ARect.Equals(BRect) Then
    Writeln ( 'ARect equals BRect' )
  Else
    Writeln ( 'ARect does not equal BRect !' );
end.
```

TRect.Assign

Declaration: Procedure Trect.Assign (XA, YA, XB, YB: Sw_Integer);

Description: Assign sets the corner points of the rectangle to (XA,YA) and (Xb,Yb).

Errors: None.

See also: Copy ([319](#))

For an example, see TRect.Copy ([319](#)).

17.5 TObject

The full declaration of the TObject type is:

```
TYPE
  TObject = OBJECT
    CONSTRUCTOR Init;
    PROCEDURE Free;
    DESTRUCTOR Done;Virtual;
  END;
  PObject = ^TObject;
```

TObject.Init

Declaration: Constructor TObject.Init;

Description: Instantiates a new object of type TObject. It fills the instance up with Zero bytes.

Errors: None.

See also: Free ([322](#)), Done ([323](#))

For an example, see Free ([322](#))

TObject.Free

Declaration: Procedure TObject.Free;

Description: `Free` calls the destructor of the object, and releases the memory occupied by the instance of the object.

Errors: No checking is performed to see whether `self` is `nil` and whether the object is indeed allocated on the heap.

See also: `Init` ([322](#)), `Done` ([323](#))

```
program ex7;

  { Program to demonstrate the TObject.Free call }

Uses Objects;

Var O : PObject;

begin
  Writeln ( 'Memavail : ', Memavail );
  // Allocate memory for object.
  O:=New(PObject, Init);
  Writeln ( 'Memavail : ', Memavail );
  // Free memory of object.
  O^.free;
  Writeln ( 'Memavail : ', Memavail );
end.
```

TObject.Done

Declaration: `Destructor TObject.Done;Virtual;`

Description: `Done`, the destructor of `TObject` does nothing. It is mainly intended to be used in the `TObject.Free` ([322](#)) method.

The destructore `Done` does not free the memory occupied by the object.

Errors: None.

See also: `Free` ([322](#)), `Init` ([322](#))

```
program ex8;

  { Program to demonstrate the TObject.Done call }

Uses Objects;

Var O : PObject;

begin
  Writeln ( 'Memavail : ', Memavail );
  // Allocate memory for object.
  O:=New(PObject, Init);
  Writeln ( 'Memavail : ', Memavail );
  O^.Done;
  Writeln ( 'Memavail : ', Memavail );
end.
```

17.6 TStream

The TStream object is the ancestor for all streaming objects, i.e. objects that have the capability to store and retrieve data.

It defines a number of methods that are common to all objects that implement streaming, many of them are virtual, and are only implemented in the descendant types.

Programs should not instantiate objects of type TStream directly, but instead instantiate a descendant type, such as TDosStream, TMemoryStream.

This is the full declaration of the TStream object:

```
TYPE
  TStream = OBJECT (TObject)
    Status      : Integer; { Stream status }
    ErrorInfo   : Integer; { Stream error info }
    StreamSize  : LongInt; { Stream current size }
    Position    : LongInt; { Current position }
    FUNCTION Get: PObject;
    FUNCTION StrRead: PChar;
    FUNCTION GetPos: Longint; Virtual;
    FUNCTION GetSize: Longint; Virtual;
    FUNCTION ReadStr: PString;
    PROCEDURE Open (OpenMode: Word); Virtual;
    PROCEDURE Close; Virtual;
    PROCEDURE Reset;
    PROCEDURE Flush; Virtual;
    PROCEDURE Truncate; Virtual;
    PROCEDURE Put (P: PObject);
    PROCEDURE StrWrite (P: PChar);
    PROCEDURE WriteStr (P: PString);
    PROCEDURE Seek (Pos: LongInt); Virtual;
    PROCEDURE Error (Code, Info: Integer); Virtual;
    PROCEDURE Read (Var Buf; Count: Sw_Word); Virtual;
    PROCEDURE Write (Var Buf; Count: Sw_Word); Virtual;
    PROCEDURE CopyFrom (Var S: TStream; Count: Longint);
  END;
  PStream = ^TStream;
```

TStream.Get

Declaration: Function TStream.Get : PObject;

Description: Get reads an object definition from a stream, and returns a pointer to an instance of this object.

Errors: On error, TStream.Status is set, and NIL is returned.

See also: Put ([328](#))

Program ex9;

{ Program to demonstrate TStream.Get and TStream.Put }

Uses Objects, MyObject; *{ Definition and registration of TMyObject }*

Var Obj : PMyObject;

```
S : PStream;

begin
  Obj:=New(PMyObject, Init);
  Obj^.SetField($1111);
  Writeln ('Field value : ',Obj^.GetField);
  { Since Stream is an abstract type, we instantiate a TMemoryStream }
  S:=New(PMemoryStream, Init(100,10));
  S^.Put(Obj);
  Writeln ('Disposing object');
  S^.Seek(0);
  Dispose(Obj, Done);
  Writeln ('Reading object');
  Obj:=PMyObject(S^.Get);
  Writeln ('Field Value : ',Obj^.GetField);
  Dispose(Obj, Done);
end.
```

TStream.StrRead

Declaration: `Function TStream.StrRead: PChar;`

Description: `StrRead` reads a string from the stream, allocates memory for it, and returns a pointer to a null-terminated copy of the string on the heap.

Errors: On error, `Nil` is returned.

See also: `StrWrite` ([329](#)), `ReadStr` ([326](#))

```
Program ex10;

{
  Program to demonstrate the TStream.StrRead TStream.StrWrite functions
}

Uses objects;

Var P : PChar;
    S : PStream;

begin
  P:='Constant Pchar string';
  Writeln ('Writing to stream : "',P,'"');
  S:=New(PMemoryStream, Init(100,10));
  S^.StrWrite(P);
  S^.Seek(0);
  P:=Nil;
  P:=S^.StrRead;
  Dispose (S, Done);
  Writeln ('Read from stream : "',P,'"');
  Freemem(P, Strlen(P)+1);
end.
```

TStream.GetPos

Declaration: `TStream.GetPos : Longint; Virtual;`

Description: If the stream's status is `stOk`, `GetPos` returns the current position in the stream. Otherwise it returns `-1`

Errors: `-1` is returned if the status is an error condition.

See also: [Seek \(329\)](#), [GetSize \(326\)](#)

```
Program ex11;

{ Program to demonstrate the TStream.GetPos function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L := 'Some kind of string';
  S := New(PMemoryStream, Init(100,10));
  Writeln ('Stream position before write : ', S^.GetPos);
  S^.WriteStr(@L);
  Writeln ('Stream position after write : ', S^.GetPos);
  Dispose(S, Done);
end.
```

TStream.GetSize

Declaration: `Function TStream.GetSize: Longint; Virtual;`

Description: If the stream's status is `stOk` then `GetSize` returns the size of the stream, otherwise it returns `-1`.

Errors: `-1` is returned if the status is an error condition.

See also: [Seek \(329\)](#), [GetPos \(325\)](#)

```
Program ex12;

{ Program to demonstrate the TStream.GetSize function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L := 'Some kind of string';
  S := New(PMemoryStream, Init(100,10));
  Writeln ('Stream size before write : ', S^.GetSize);
  S^.WriteStr(@L);
  Writeln ('Stream size after write : ', S^.GetSize);
  Dispose(S, Done);
end.
```

TStream.ReadStr

Declaration: `Function TStream.ReadStr: PString;`

Description: `ReadStr` reads a string from the stream, copies it to the heap and returns a pointer to this copy. The string is saved as a pascal string, and hence is NOT null terminated.

Errors: On error (e.g. not enough memory), `Nil` is returned.

See also: `StrRead` ([325](#))

```
Program ex13;  
  
{  
  Program to demonstrate the TStream.ReadStr TStream.WriteStr functions  
}  
  
Uses objects;  
  
Var P : PString;  
      L : String;  
      S : PStream;  
  
begin  
  L := 'Constant string line';  
  Writeln ( 'Writing to stream : '' ,L, '' ' );  
  S := New(PMemoryStream, Init(100,10));  
  S^.WriteStr(@L);  
  S^.Seek(0);  
  P := S^.ReadStr;  
  L := P^;  
  DisposeStr(P);  
  Dispose (S, Done);  
  Writeln ( 'Read from stream : '' ,L, '' ' );  
end.
```

TStream.Open

Declaration: `Procedure TStream.Open (OpenMode: Word); Virtual;`

Description: `Open` is an abstract method, that should be overridden by descendent objects. Since opening a stream depends on the stream's type this is not surprising.

Errors: None.

See also: `Close` ([327](#)), `Reset` ([328](#))

For an example, see `TDosStream.Open` ([334](#)).

TStream.Close

Declaration: `Procedure TStream.Close; Virtual;`

Description: `Close` is an abstract method, that should be overridden by descendent objects. Since Closing a stream depends on the stream's type this is not surprising.

Errors: None.

See also: `Open` ([327](#)), `Reset` ([328](#))

for an example, see `TDosStream.Open` ([334](#)).

TStream.Reset

Declaration: `PROCEDURE TStream.Reset ;`

Description: `Reset` sets the stream's status to 0, as well as the `ErrorInfo`

Errors: None.

See also: `Open` (327), `Close` (327)

TStream.Flush

Declaration: `Procedure TStream.Flush; Virtual;`

Description: `Flush` is an abstract method that should be overridden by descendent objects. It serves to enable the programmer to tell streams that implement a buffer to clear the buffer.

Errors: None.

See also: `Truncate` (328)

for an example, see `TBufStream.Flush` (337).

TStream.Truncate

Declaration: `Procedure TStream.Truncate; Virtual;`

Description: `Truncate` is an abstract procedure that should be overridden by descendent objects. It serves to enable the programmer to truncate the size of the stream to the current file position.

Errors: None.

See also: `Seek` (329)

For an example, see `TDosStream.Truncate` (333).

TStream.Put

Declaration: `Procedure TStream.Put (P: PObject);`

Description: `Put` writes the object pointed to by `P`. `P` should be non-nil. The object type must have been registered with `RegisterType` (315).

After the object has been written, it can be read again with `Get` (324).

Errors: No check is done whether `P` is `Nil` or not. Passing `Nil` will cause a run-time error 216 to be generated. If the object has not been registered, the status of the stream will be set to `stPutError`.

See also: `Get` (324)

For an example, see `TStream.Get` (324);

TStream.StrWrite

Declaration: `Procedure TStream.StrWrite (P: PChar);`

Description: `StrWrite` writes the null-terminated string `P` to the stream. `P` can only be 65355 bytes long.

Errors: None.

See also: [WriteStr \(329\)](#), [StrRead \(325\)](#), [ReadStr \(326\)](#)

For an example, see [TStream.StrRead \(325\)](#).

TStream.WriteString

Declaration: `Procedure TStream.WriteString (P: PString);`

Description: `StrWrite` writes the pascal string pointed to by `P` to the stream.

Errors: None.

See also: [StrWrite \(329\)](#), [StrRead \(325\)](#), [ReadStr \(326\)](#)

For an example, see [TStream.ReadStr \(326\)](#).

TStream.Seek

Declaration: `PROCEDURE TStream.Seek (Pos: LongInt); Virtual;`

Description: `Seek` sets the position to `Pos`. This position is counted from the beginning, and is zero based. (i.e. `seek(0)` sets the position pointer on the first byte of the stream)

Errors: If `Pos` is larger than the stream size, `Status` is set to `StSeekError`.

See also: [GetPos \(325\)](#), [GetSize \(326\)](#)

For an example, see [TDosStream.Seek \(333\)](#).

TStream.Error

Declaration: `Procedure TStream.Error (Code, Info: Integer); Virtual;`

Description: `Error` sets the stream's status to `Code` and `ErrorInfo` to `Info`. If the `StreamError` procedural variable is set, `Error` executes it, passing `Self` as an argument.

This method should not be called directly from a program. It is intended to be used in descendent objects.

Errors: None.

See also:

TStream.Read

Declaration: `Procedure TStream.Read (Var Buf; Count: Sw_Word); Virtual;`

Description: Read is an abstract method that should be overridden by descendent objects.

Read reads Count bytes from the stream into Buf. It updates the position pointer, increasing it's value with Count. Buf must be large enough to contain Count bytes.

Errors: No checking is done to see if Buf is large enough to contain Count bytes.

See also: Write (330), ReadStr (326), StrRead (325)

```
program ex18;

{ Program to demonstrate the TStream.Read method }

Uses Objects;

Var Buf1, Buf2 : Array[1..1000] of Byte;
    I : longint;
    S : PMemoryStream;

begin
  For I:=1 to 1000 do
    Buf1[I]:=Random(1000);
  Buf2:=Buf1;
  S:=New(PMemoryStream, Init(100,10));
  S^.Write(Buf1, SizeOf(Buf1));
  S^.Seek(0);
  For I:=1 to 1000 do
    Buf1[I]:=0;
  S^.Read(Buf1, SizeOf(Buf1));
  For I:=1 to 1000 do
    If Buf1[I]<>Buf2[I] then
      Writeln('Buffer differs at position ', I);
  Dispose(S, Done);
end.
```

TStream.Write

Declaration: `Procedure TStream.Write (Var Buf; Count: Sw_Word); Virtual;`

Description: Write is an abstract method that should be overridden by descendent objects.

Write writes Count bytes to the stream from Buf. It updates the position pointer, increasing it's value with Count.

Errors: No checking is done to see if Buf actually contains Count bytes.

See also: Read (330), WriteStr (329), StrWrite (329)

For an example, see TStream.Read (330).

TStream.CopyFrom

Declaration: `Procedure TStream.CopyFrom (Var S: TStream; Count: Longint);`

Description: `CopyFrom` reads `Count` bytes from stream `S` and stores them in the current stream. It uses the `Read` (330) method to read the data, and the `Write` (330) method to write in the current stream.

Errors: None.

See also: `Read` (330), `Write` (330)

```
Program ex19;

{ Program to demonstrate the TStream.CopyFrom function }

Uses objects;

Var P : PString;
    L : String;
    S1,S2 : PStream;

begin
  L:= 'Constant string line';
  Writeln ( 'Writing to stream 1 : '' ,L, '' ');
  S1:=New(PMemoryStream, Init(100,10));
  S2:=New(PMemoryStream, Init(100,10));
  S1^.WriteStr(@L);
  S1^.Seek(0);
  Writeln ( 'Copying contents of stream 1 to stream 2' );
  S2^.Copyfrom(S1^,S1^.GetSize);
  S2^.Seek(0);
  P:=S2^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Dispose (S1,Done);
  Dispose (S2,Done);
  Writeln ( 'Read from stream 2 : '' ,L, '' ');
end.
```

17.7 TDosStream

`TDosStream` is a stream that stores it's contents in a file. it overrides a couple of methods of `TStream` for this.

In addition to the fields inherited from `TStream` (see section 17.6, page 324), there are some extra fields, that describe the file. (mainly the name and the OS file handle)

No buffering in memory is done when using `TDosStream`. All data are written directly to the file. For a stream that buffers in memory, see section 17.8, page 335.

Here is the full declaration of the `TDosStream` object:

```
TYPE
  TDosStream = OBJECT (TStream)
    Handle: THandle; { DOS file handle }
    FName : AsciiZ; { AsciiZ filename }
    CONSTRUCTOR Init (FileName: FNameStr; Mode: Word);
    DESTRUCTOR Done; Virtual;
    PROCEDURE Close; Virtual;
    PROCEDURE Truncate; Virtual;
    PROCEDURE Seek (Pos: LongInt); Virtual;
```

```
PROCEDURE Open (OpenMode: Word); Virtual;  
PROCEDURE Read (Var Buf; Count: Sw_Word); Virtual;  
PROCEDURE Write (Var Buf; Count: Sw_Word); Virtual;  
END;  
PDosStream = ^TDosStream;
```

TDosStream.Init

Declaration: Constructor Init (FileName: FNameStr; Mode: Word);

Description: Init instantiates an instance of TDosStream. The name of the file that contains (or will contain) the data of the stream is given in FileName. The Mode parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreateCreates a new file.

stOpenReadRead access only.

stOpenWriteWrite access only.

stOpenReadRead and write access.

Errors: On error, Status is set to stInitError, and ErrorInfo is set to the DOS error code.

See also: Done ([332](#))

For an example, see TDosStream.Truncate ([333](#)).

TDosStream.Done

Declaration: Destructor TDosStream.Done; Virtual;

Description: Done closes the file if it was open and cleans up the instance of TDosStream.

Errors: None.

See also: Init ([332](#)), Close ([332](#))

for an example, see e.g. TDosStream.Truncate ([333](#)).

TDosStream.Close

Declaration: Procedure TDosStream.Close; Virtual;

Description: Close closes the file if it was open, and sets Handle to -1. Contrary to Done ([332](#)) it does not clean up the instance of TDosStream

Errors: None.

See also: TStream.Close ([327](#)), Init ([332](#)), Done ([332](#))

For an example, see TDosStream.Open ([334](#)).

TDosStream.Truncate

Declaration: Procedure TDosStream.Truncate; Virtual;

Description: If the status of the stream is stOK, then Truncate tries to truncate the stream size to the current file position.

Errors: If an error occurs, the stream's status is set to stError and ErrorInfo is set to the OS error code.

See also: TStream.Truncate ([328](#)), GetSize ([326](#))

```
Program ex16;

{ Program to demonstrate the TStream.Truncate method }

Uses Objects;

Var L : String;
    P : PString;
    S : PDosStream; { Only one with Truncate implemented. }

begin
  L:= 'Some constant string';
  { Buffer size of 100 }
  S:=New(PDosStream, Init('test.dat', stcreate));
  Writeln ('Writing "', L, '" to stream with handle ', S^.Handle);
  S^.WriteStr(@L);
  S^.WriteStr(@L);
  { Close calls flush first }
  S^.Close;
  S^.Open (stOpen);
  Writeln ('Size of stream is : ', S^.GetSize);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "', L, '" from stream with handle ', S^.Handle);
  S^.Truncate;
  Writeln ('Truncated stream. Size is : ', S^.GetSize);
  S^.Close;
  Dispose (S, Done);
end.
```

TDosStream.Seek

Declaration: Procedure TDosStream.Seek (Pos: LongInt); Virtual;

Description: If the stream's status is stOK, then Seek sets the file position to Pos. Pos is a zero-based offset, counted from the beginning of the file.

Errors: In case an error occurs, the stream's status is set to stSeekError, and the OS error code is stored in ErrorInfo.

See also: TStream.Seek ([329](#)), GetPos ([325](#))

```
Program ex17;

{ Program to demonstrate the TStream.Seek method }
```

Uses Objects;

```
Var L : String;
      Marker : Word;
      P : PString;
      S : PDosStream;

begin
  L:= 'Some constant string';
  { Buffer size of 100 }
  S:=New(PDosStream, Init( 'test.dat', stcreate ));
  Writeln ( 'Writing "', L, '" to stream.' );
  S^. WriteStr (@L);
  Marker:=S^. GetPos;
  Writeln ( 'Set marker at ', Marker);
  L:= 'Some other constant String';
  Writeln ( 'Writing "', L, '" to stream.' );
  S^. WriteStr (@L);
  S^. Close;
  S^. Open ( stOpenRead );
  Writeln ( 'Size of stream is : ', S^. GetSize );
  Writeln ( 'Seeking to marker' );
  S^. Seek (Marker);
  P:=S^. ReadStr;
  L:=P^;
  DisposeStr (P);
  Writeln ( 'Read "', L, '" from stream.' );
  S^. Close;
  Dispose (S, Done);
end.
```

TDosStream.Open

Declaration: `Procedure TDosStream.Open (OpenMode: Word); Virtual;`

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a `Close` (332) call.

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: `TStream.Open` (327), `Close` (332)

Program ex14;

{ Program to demonstrate the TStream.Close method }

Uses Objects;

```
Var L : String;
      P : PString;
      S : PDosStream; { Only one with Close implemented. }
```

```
begin
  L:= 'Some constant string';
  S:=New(PDosStream, Init( 'test.dat', stcreate ));
```



```
Writeln ( 'Writing "',L,'" to stream with handle ',S^.Handle);  
S^.WriteStr(@L);  
S^.Close;  
Writeln ( 'Closed stream. File handle is ',S^.Handle);  
S^.Open (stOpenRead);  
P:=S^.ReadStr;  
L:=P^;  
DisposeStr(P);  
Writeln ( 'Read "',L,'" from stream with handle ',S^.Handle);  
S^.Close;  
Dispose (S,Done);  
end.
```

TDosStream.Read

Declaration: `Procedure TDosStream.Read (Var Buf; Count: Sw_Word); Virtual;`

Description: If the Stream is open and the stream status is `stOK` then Read will read Count bytes from the stream and place them in Buf.

Errors: In case of an error, Status is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: [TStream.Read \(330\)](#), [Write \(335\)](#)

For an example, see [TStream.Read \(330\)](#).

TDosStream.Write

Declaration: `Procedure TDosStream.Write (Var Buf; Count: Sw_Word); Virtual;`

Description: If the Stream is open and the stream status is `stOK` then Write will write Count bytes from Buf and place them in the stream.

Errors: In case of an error, Status is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: [TStream.Write \(330\)](#), [Read \(335\)](#)

For an example, see [TStream.Read \(330\)](#).

17.8 TBufStream

Bufstream implements a buffered file stream. That is, all data written to the stream is written to memory first. Only when the buffer is full, or on explicit request, the data is written to disk.

Also, when reading from the stream, first the buffer is checked if there is any unread data in it. If so, this is read first. If not the buffer is filled again, and then the data is read from the buffer.

The size of the buffer is fixed and is set when constructing the file.

This is useful if you need heavy throughput for your stream, because it speeds up operations.

TYPE

```
TBufStream = OBJECT (TDosStream)  
    LastMode: Byte;      { Last buffer mode }
```

```
    BufSize : Sw_Word;      { Buffer size }
    BufPtr   : Sw_Word;      { Buffer start }
    BufEnd   : Sw_Word;      { Buffer end }
    Buffer    : PByteArray;   { Buffer allocated }
CONSTRUCTOR Init (FileName: FNameStr; Mode, Size: Word);
DESTRUCTOR Done; Virtual;
PROCEDURE Close; Virtual;
PROCEDURE Flush; Virtual;
PROCEDURE Truncate; Virtual;
PROCEDURE Seek (Pos: LongInt); Virtual;
PROCEDURE Open (OpenMode: Word); Virtual;
PROCEDURE Read (Var Buf; Count: Sw_Word); Virtual;
PROCEDURE Write (Var Buf; Count: Sw_Word); Virtual;
END;
PBufStream = ^TBufStream;
```

TBufStream.Init

Declaration: Constructor Init (FileName: FNameStr; Mode,Size: Word);

Description: Init instantiates an instance of TBufStream. The name of the file that contains (or will contain) the data of the stream is given in FileName. The Mode parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreateCreates a new file.

stOpenReadRead access only.

stOpenWriteWrite access only.

stOpenReadRead and write access.

The Size parameter determines the size of the buffer that will be created. It should be different from zero.

Errors: On error, Status is set to stInitError, and ErrorInfo is set to the DOS error code.

See also: TDosStream.Init ([332](#)), Done ([336](#))

For an example see TBufStream.Flush ([337](#)).

TBufStream.Done

Declaration: Destructor TBufStream.Done; Virtual;

Description: Done flushes and closes the file if it was open and cleans up the instance of TBufStream.

Errors: None.

See also: TDosStream.Done ([332](#)), Init ([336](#)), Close ([337](#))

For an example see TBufStream.Flush ([337](#)).

TBufStream.Close

Declaration: `Procedure TBufStream.Close; Virtual;`

Description: `Close` flushes and closes the file if it was open, and sets `Handle` to -1. Contrary to `Done` (336) it does not clean up the instance of `TBufStream`

Errors: None.

See also: `TStream.Close` (327), `Init` (336), `Done` (336)

For an example see `TBufStream.Flush` (337).

TBufStream.Flush

Declaration: `Procedure TBufStream.Flush; Virtual;`

Description: When the stream is in write mode, the contents of the buffer are written to disk, and the buffer position is set to zero.

When the stream is in read mode, the buffer position is set to zero.

Errors: Write errors may occur if the file was in write mode. see `Write` (339) for more info on the errors.

See also: `TStream.Close` (327), `Init` (336), `Done` (336)

Program ex15;

{ Program to demonstrate the TStream.Flush method }

Uses Objects;

Var L : **String**;
P : **PString**;
S : **PBufStream**; *{ Only one with Flush implemented. }*

begin

```
L:= 'Some constant string';  
{ Buffer size of 100 }  
S:=New(PBufStream, Init('test.dat', stcreate, 100));  
Writeln('Writing "', L, '" to stream with handle ', S^.Handle);  
S^.WriteStr(@L);  
{ At this moment, there is no data on disk yet. }  
S^.Flush;  
{ Now there is. }  
S^.WriteStr(@L);  
{ Close calls flush first }  
S^.Close;  
Writeln('Closed stream. File handle is ', S^.Handle);  
S^.Open(stOpenRead);  
P:=S^.ReadStr;  
L:=P^;  
DisposeStr(P);  
Writeln('Read "', L, '" from stream with handle ', S^.Handle);  
S^.Close;  
Dispose(S, Done);
```

end.

TBufStream.Truncate

Declaration: `Procedure TBufStream.Truncate; Virtual;`

Description: If the status of the stream is `stOK`, then `Truncate` tries to flush the buffer, and then truncates the stream size to the current file position.

Errors: Errors can be those of `Flush` (337) or `TDosStream.Truncate` (333).

See also: `TStream.Truncate` (328), `TDosStream.Truncate` (333), `GetSize` (326)

For an example, see `TDosStream.Truncate` (333).

TBufStream.Seek

Declaration: `Procedure TBufStream.Seek (Pos: LongInt); Virtual;`

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: `TStream.Seek` (329), `GetPos` (325)

For an example, see `TStream.Seek` (329);

TBufStream.Open

Declaration: `Procedure TBufStream.Open (OpenMode: Word); Virtual;`

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a `Close` (337) call.

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: `TStream.Open` (327), `Close` (337)

For an example, see `TDosStream.Open` (334).

TBufStream.Read

Declaration: `Procedure TBufStream.Read (Var Buf; Count: Sw_Word); Virtual;`

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

`Read` will first try to read the data from the stream's internal buffer. If insufficient data is available, the buffer will be filled before continuing to read. This process is repeated until all needed data has been read.

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: `TStream.Read` (330), `Write` (339)

For an example, see `TStream.Read` (330).

TBufStream.Write

Declaration: Procedure TBufStream.Write (Var Buf; Count: Sw_Word); Virtual;

Description: If the Stream is open and the stream status is stOK then Write will write Count bytes from Buf and place them in the stream.

Write will first try to write the data to the stream's internal buffer. When the internal buffer is full, then the contents will be written to disk. This process is repeated until all data has been written.

Errors: In case of an error, Status is set to StWriteError, and ErrorInfo gets the OS specific error.

See also: TStream.Write ([330](#)), Read ([338](#))

For an example, see TStream.Read ([330](#)).

17.9 TMemoryStream

The TMemoryStream object implements a stream that stores its data in memory. The data is stored on the heap, with the possibility to specify the maximum amount of data, and the size of the memory blocks being used.

```
TYPE
  TMemoryStream = OBJECT (TStream)
    BlkCount: Sw_Word;      { Number of segments }
    BlkSize : Word;         { Memory block size  }
    MemSize : LongInt;      { Memory alloc size  }
    BlkList : PPointerArray; { Memory block list }
    CONSTRUCTOR Init (ALimit: Longint; ABlockSize: Word);
    DESTRUCTOR Done;
    PROCEDURE Truncate;
    PROCEDURE Read (Var Buf; Count: Sw_Word);
    PROCEDURE Write (Var Buf; Count: Sw_Word);
  END;
  PMemoryStream = ^TMemoryStream;
```

TMemoryStream.Init

Declaration: Constructor TMemoryStream.Init (ALimit: Longint; ABlockSize: Word);

Description: Init instantiates a new TMemoryStream object. The memorystreamobject will initially allocate at least ALimit bytes memory, divided into memory blocks of size ABlockSize. The number of blocks needed to get to ALimit bytes is rounded up.

By default, the number of blocks is 1, and the size of a block is 8192. This is selected if you specify 0 as the blocksize.

Errors: If the stream cannot allocate the initial memory needed for the memory blocks, then the stream's status is set to stInitError.

See also: Done ([340](#))

For an example, see e.g TStream.CopyFrom ([330](#)).

TMemoryStream.Done

Declaration: Destructor TMemoryStream.Done; Virtual;

Description: Done releases the memory blocks used by the stream, and then cleans up the memory used by the stream object itself.

Errors: None.

See also: Init ([339](#))

For an example, see e.g TStream.CopyFrom ([330](#)).

TMemoryStream.Truncate

Declaration: Procedure TMemoryStream.Truncate; Virtual;

Description: Truncate sets the size of the memory stream equal to the current position. It de-allocates any memory-blocks that are no longer needed, so that the new size of the stream is the current position in the stream, rounded up to the first multiple of the stream blocksize.

Errors: If an error occurs during memory de-allocation, the stream's status is set to stError

See also: TStream.Truncate ([328](#))

```
Program ex20;

{ Program to demonstrate the TMemoryStream.Truncate method }

Uses Objects;

Var L : String;
    P : PString;
    S : PMemoryStream;
    I, InitMem : Longint;

begin
  InitMem:=Memavail;
  L:= 'Some constant string';
  { Buffer size of 100 }
  S:=New(PMemoryStream, Init(1000,100));
  Writeln ( 'Free memory : ', Memavail);
  Writeln ( 'Writing 100 times "', L, '" to stream. ');
  For I:=1 to 100 do
    S^.WriteStr(@L);
  Writeln ( 'Finished. Free memory : ', Memavail);
  S^.Seek(100);
  S^.Truncate;
  Writeln ( 'Truncated at byte 100. Free memory : ', Memavail);
  Dispose (S, Done);
  Writeln ( 'Finished. Lost ', InitMem-Memavail, ' Bytes. ');
end.
```

TMemoryStream.Read

Declaration: Procedure Read (Var Buf; Count: Sw_Word); Virtual;

Description: Read reads Count bytes from the stream to Buf. It updates the position of the stream.

Errors: If there is not enough data available, no data is read, and the stream's status is set to `stReadError`.

See also: `TStream.Read`, `Write` (341)

For an example, see `TStream.Read` (330).

TMemoryStream.Write

Declaration: `Procedure Write (Var Buf; Count: Sw_Word); Virtual;`

Description: Write copies `Count` bytes from `Buf` to the stream. It updates the position of the stream.

If not enough memory is available to hold the extra `Count` bytes, then the stream will try to expand, by allocating as much blocks with size `BlkSize` (as specified in the constructor call `Init` (339)) as needed.

Errors: If the stream cannot allocate more memory, then the status is set to `stWriteError`

See also: `TStream.Write` (330), `Read` (340)

For an example, see `TStream.Read` (330).

17.10 TCollection

The `TCollection` object manages a collection of pointers or objects. It also provides a series of methods to manipulate these pointers or objects.

Whether or not objects are used depends on the kind of calls you use. All kinds come in 2 flavors, one for objects, one for pointers.

This is the full declaration of the `TCollection` object:

TYPE

```
TItemList = Array [0..MaxCollectionSize - 1] Of Pointer;  
PItemList = ^TItemList;
```

```
TCollection = OBJECT (TObject)  
  Items: PItemList; { Item list pointer }  
  Count: Sw_Integer; { Item count }  
  Limit: Sw_Integer; { Item limit count }  
  Delta: Sw_Integer; { Inc delta size }  
  Constructor Init (ALimit, ADelta: Sw_Integer);  
  Constructor Load (Var S: TStream);  
  Destructor Done; Virtual;  
  Function At (Index: Sw_Integer): Pointer;  
  Function IndexOf (Item: Pointer): Sw_Integer; Virtual;  
  Function GetItem (Var S: TStream): Pointer; Virtual;  
  Function LastThat (Test: Pointer): Pointer;  
  Function FirstThat (Test: Pointer): Pointer;  
  Procedure Pack;  
  Procedure FreeAll;  
  Procedure DeleteAll;  
  Procedure Free (Item: Pointer);  
  Procedure Insert (Item: Pointer); Virtual;  
  Procedure Delete (Item: Pointer);
```

```
Procedure AtFree (Index: Sw_Integer);
Procedure FreeItem (Item: Pointer); Virtual;
Procedure AtDelete (Index: Sw_Integer);
Procedure ForEach (Action: Pointer);
Procedure SetLimit (ALimit: Sw_Integer); Virtual;
Procedure Error (Code, Info: Integer); Virtual;
Procedure AtPut (Index: Sw_Integer; Item: Pointer);
Procedure AtInsert (Index: Sw_Integer; Item: Pointer);
Procedure Store (Var S: TStream);
Procedure PutItem (Var S: TStream; Item: Pointer); Virtual;
END;
PCollection = ^TCollection;
```

TCollection.Init

Declaration: Constructor TCollection.Init (ALimit, ADelta: Sw_Integer);

Description: Init initializes a new instance of a collection. It sets the (initial) maximum number of items in the collection to ALimit. ADelta is the increase size : The number of memory places that will be allocated in case ALimit is reached, and another element is added to the collection.

Errors: None.

See also: Load ([342](#)), Done ([343](#))

For an example, see TCollection.ForEach ([352](#)).

TCollection.Load

Declaration: Constructor TCollection.Load (Var S: TStream);

Description: Load initializes a new instance of a collection. It reads from stream S the item count, the item limit count, and the increase size. After that, it reads the specified number of items from the stream.

Errors: Errors returned can be those of GetItem ([344](#)).

See also: Init ([342](#)), GetItem ([344](#)), Done ([343](#)).

Program ex22;

{ Program to demonstrate the TCollection.Load method }

Uses Objects, MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;
M : PMyObject;
I : Longint;
S : PMemoryStream;

begin

C:=New(PCollection, Init(100,10));

For I:=1 **to** 100 **do**

begin

M:=New(PMyObject, Init);

M^.SetField(100-I);

C^.Insert(M);


```
    end;  
    Writeln ( 'Inserted ',C^.Count, ' objects' );  
    S:=New(PMemoryStream, Init(1000,10));  
    C^.Store(S^);  
    C^.FreeAll;  
    Dispose(C,Done);  
    S^.Seek(0);  
    C^.Load(S^);  
    Writeln ( 'Read ',C^.Count, ' objects from stream.' );  
    Dispose(S,Done);  
    Dispose(C,Done);  
end.
```

TCollection.Done

Declaration: Destructor TCollection.Done; Virtual;

Description: Done frees all objects in the collection, and then releases all memory occupied by the instance.

Errors: None.

See also: Init ([342](#)), FreeAll ([347](#))

For an example, see TCollection.ForEach ([352](#)).

TCollection.At

Declaration: Function TCollection.At (Index: Sw_Integer): Pointer;

Description: At returns the item at position Index.

Errors: If Index is less than zero or larger than the number of items in the collection, seeplErrorTCollection.Error is called with coIndexError and Index as arguments, resulting in a run-time error.

See also: Insert ([349](#))

```
Program ex23;  
  
    { Program to demonstrate the TCollection.At method }  
  
    Uses Objects,MyObject; { For TMyObject definition and registration }  
  
    Var C : PCollection;  
        M : PMyObject;  
        I : Longint;  
  
    begin  
        C:=New(PCollection, Init(100,10));  
        For I:=1 to 100 do  
            begin  
                M:=New(PMyObject, Init);  
                M^.SetField(100-I);  
                C^.Insert(M);  
            end;  
        For I:=0 to C^.Count-1 do  
            begin  
                M:=C^.At(I);
```

```
      Writeln ('Object ', i, ' has field : ', M^.GetField);  
    end;  
    C^.FreeAll;  
    Dispose(C, Done);  
end.
```

TCollection.IndexOf

Declaration: Function TCollection.IndexOf (Item: Pointer): Sw_Integer; Virtual;

Description: IndexOf returns the index of Item in the collection. If Item isn't present in the collection, -1 is returned.

Errors:

See also:

```
Program ex24;  
  
  { Program to demonstrate the TCollection.IndexOf method }  
  
  Uses Objects, MyObject; { For TMyObject definition and registration }  
  
  Var C : PCollection;  
      M, Keep : PMyObject;  
      I : Longint;  
  
  begin  
    Randomize;  
    C:=New(PCollection, Init(100,10));  
    Keep:=Nil;  
    For I:=1 to 100 do  
      begin  
        M:=New(PMyObject, Init);  
        M^.SetField(I-1);  
        If Random<0.1 then  
          Keep:=M;  
          C^.Insert(M);  
        end;  
    If Keep=Nil then  
      begin  
        Writeln ('Please run again. No object selected');  
        Halt(1);  
      end;  
    Writeln ('Selected object has field : ', Keep^.GetField);  
    Write ('Selected object has index : ', C^.IndexOf(Keep));  
    Writeln (' should match it's field. ');  
    C^.FreeAll;  
    Dispose(C, Done);  
  end.
```

TCollection.GetItem

Declaration: Function TCollection.GetItem (Var S: TStream): Pointer; Virtual;

Description: GetItem reads a single item off the stream S, and returns a pointer to this item. This method is used internally by the Load method, and should not be used directly.

Errors: Possible errors are the ones from TStream.Get (324).

See also: TStream.Get (324), seepIStoreTCollection.Store

TCollection.LastThat

Declaration: `Function TCollection.LastThat (Test: Pointer): Pointer;`

Description: This function returns the last item in the collection for which Test returns a non-nil result. Test is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: FirstThat (345)

```
Program ex21;  
  
  { Program to demonstrate the TCollection.Foreach method }  
  
Uses Objects, MyObject; { For TMyObject definition and registration }  
  
Var C : PCollection;  
      M : PMyObject;  
      I : Longint;  
  
Function CheckField (Dummy: Pointer; P : PMyObject) : Longint;  
  
  begin  
    If P^.GetField<56 then  
      Checkfield:=1  
    else  
      CheckField:=0;  
  end;  
  
begin  
  C:=New(PCollection, Init(100,10));  
  For I:=1 to 100 do  
    begin  
      M:=New(PMyObject, Init);  
      M^.SetField(I);  
      C^.Insert(M);  
    end;  
  Writeln ('Inserted ', C^.Count, ' objects');  
  Writeln ('Last one for which Field<56 has index (should be 54) : ',  
    C^.IndexOf(C^.LastThat(@CheckField)));  
  C^.FreeAll;  
  Dispose(C, Done);  
end.
```

TCollection.FirstThat

Declaration: `Function TCollection.FirstThat (Test: Pointer): Pointer;`

Description: This function returns the first item in the collection for which Test returns a non-nil result. Test is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: LastThat ([345](#))

```
Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
    If P^.GetField>56 then
        Checkfield:=1
    else
        CheckField:=0;
end;

begin
    C:=New(PCollection,Init(100,10));
    For I:=1 to 100 do
        begin
            M:=New(PMyObject,Init);
            M^.SetField(I);
            C^.Insert(M);
        end;
    Writeln ('Inserted ',C^.Count,' objects');
    Writeln ('first one for which Field>56 has index (should be 56) : ',
        C^.IndexOf(C^.FirstThat(@CheckField)));
    C^.FreeAll;
    Dispose(C,Done);
end.
```

TCollection.Pack

Declaration: Procedure TCollection.Pack;

Description: Pack removes all Nil pointers from the collection, and adjusts Count to reflect this change. No memory is freed as a result of this call. In order to free any memory, you can call SetLimit with an argument of Count after a call to Pack.

Errors: None.

See also: SetLimit ([353](#))

```
Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;
```

```
Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField>56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection, Init(100,10));
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I);
      C^.Insert(M);
    end;
  Writeln ( 'Inserted ',C^.Count, ' objects' );
  Writeln ( 'first one for which Field>56 has index (should be 56) : ',
    C^.IndexOf(C^.FirstThat (@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.
```

TCollection.FreeAll

Declaration: `Procedure TCollection.FreeAll;`

Description: `FreeAll` calls the destructor of each object in the collection. It doesn't release any memory occupied by the collection itself, but it does set `Count` to zero.

Errors:

See also: `DeleteAll` ([348](#)), `FreeItem` ([351](#))

```
Program ex28;

{ Program to demonstrate the TCollection.FreeAll method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I,InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln ( 'Initial memory : ',InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
  Writeln ( 'Added 100 Items. Memory available : ',Memavail);
```

```
Write ('Lost : ', Initmem-Memavail, ' bytes. ');
Write  ('( Should be 100*', SizeOf(TMyObject));
Writeln ('=', 100*SizeOf(TMyObject), ') ');
C^.FreeAll;
Writeln ('Freed all objects. Memory available : ', Memavail);
Writeln ('Lost : ', Initmem-Memavail, ' bytes. ');
Dispose(C, Done);
end.
```

TCollection.DeleteAll

Declaration: Procedure TCollection.DeleteAll;

Description: DeleteAll deletes all elements from the collection. It just sets the Count variable to zero. Contrary to FreeAll (347), DeleteAll doesn't call the destructor of the objects.

Errors: None.

See also: FreeAll (347), Delete (349)

```
Program ex29;

{
  Program to demonstrate the TCollection.DeleteAll method
  Compare with example 28, where FreeAll is used.
}

Uses Objects, MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I, InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln ('Initial memory : ', InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
  Writeln ('Added 100 Items. Memory available : ', Memavail);
  Write ('Lost : ', Initmem-Memavail, ' bytes. ');
  Write  ('( Should be 100*', SizeOf(TMyObject));
  Writeln ('=', 100*SizeOf(TMyObject), ') ');
  C^.DeleteAll;
  Writeln ('Deleted all objects. Memory available : ', Memavail);
  Writeln ('Lost : ', Initmem-Memavail, ' bytes. ');
  Dispose(C, Done);
end.
```

TCollection.Free

Declaration: Procedure TCollection.Free (Item: Pointer);

Description: `Free` Deletes Item from the collection, and calls the destructor `Done` of the object.

Errors: If the Item is not in the collection, Error will be called with `coIndexError`.

See also: `FreeItem` ([351](#)),

```
Program ex30;

{ Program to demonstrate the TCollection.Free method }

Uses Objects, MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I, InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  WriteLn ('Initial memory : ', InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
  WriteLn ('Added 100 Items. Memory available : ', Memavail);
  Write ('Lost : ', Initmem-Memavail, ' bytes. ');
  Write ('( Should be 100*', SizeOf(TMyObject));
  WriteLn ('=', 100*SizeOf(TMyObject), ') ');
  With C^ do
    While Count>0 do Free(At(Count-1));
  WriteLn ('Freed all objects. Memory available : ', Memavail);
  WriteLn ('Lost : ', Initmem-Memavail, ' bytes. ');
  Dispose(C, Done);
end.
```

TCollection.Insert

Declaration: `Procedure TCollection.Insert (Item: Pointer); Virtual;`

Description: `Insert` inserts Item in the collection. `TCollection` inserts this item at the end, but descendent objects may insert it at another place.

Errors: None.

See also: `AtInsert` ([353](#)), `AtPut` ([353](#)),

TCollection.Delete

Declaration: `Procedure TCollection.Delete (Item: Pointer);`

Description: `Delete` deletes Item from the collection. It doesn't call the item's destructor, though. For this the `Free` ([348](#)) call is provided.

Errors: If the Item is not in the collection, Error will be called with `coIndexError`.

See also: [AtDelete \(351\)](#), [Free \(348\)](#)

```
Program ex31;

{ Program to demonstrate the TCollection.Delete method }

Uses Objects, MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I, InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln ('Initial memory : ', InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
  Writeln ('Added 100 Items. Memory available : ', Memavail);
  Write ('Lost : ', Initmem-Memavail, ' bytes. ');
  Write ('(Should be 100*', SizeOf(TMyObject));
  Writeln ('=', 100*SizeOf(TMyObject), ') ');
  With C^ do
    While Count>0 do Delete(At(Count-1));
  Writeln ('Freed all objects. Memory available : ', Memavail);
  Writeln ('Lost : ', Initmem-Memavail, ' bytes. ');
  Dispose(C, Done);
end.
```

TCollection.AtFree

Declaration: Procedure TCollection.AtFree (Index: Sw_Integer);

Description: AtFree deletes the item at position Index in the collection, and calls the item's destructor if it is not Nil.

Errors: If Index isn't valid then [Error \(353\)](#) is called with CoIndexError.

See also: [Free \(348\)](#), [AtDelete \(351\)](#)

```
Program ex32;

{ Program to demonstrate the TCollection.AtFree method }

Uses Objects, MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I, InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
```



```
InitMem:=Memavail;  
Writeln ( 'Initial memory : ',InitMem);  
For I:=1 to 100 do  
begin  
  M:=New(PMyObject, Init);  
  M^.SetField(I-1);  
  C^.Insert(M);  
end;  
Writeln ( 'Added 100 Items. Memory available : ',Memavail);  
Write ( 'Lost : ',Initmem-Memavail, ' bytes. ');  
Write ( '( Should be 100*',SizeOf(TMyObject));  
Writeln ( '= ',100*SizeOf(TMyObject), ' ) ');  
With C^ do  
  While Count>0 do AtFree(Count-1);  
Writeln ( 'Freed all objects. Memory available : ',Memavail);  
Writeln ( 'Lost : ',Initmem-Memavail, ' bytes. ');  
Dispose(C,Done);  
end.
```

TCollection.FreeItem

Declaration: Procedure TCollection.FreeItem (Item: Pointer); Virtual;

Description: FreeItem calls the destructor of Item if it is not nil.

This function is used internally by the TCollection object, and should not be called directly.

Errors: None.

See also: Free ([350](#)), seeplAtFreeTCollection.AtFree

TCollection.AtDelete

Declaration: Procedure TCollection.AtDelete (Index: Sw_Integer);

Description: AtDelete deletes the pointer at position Index in the collection. It doesn't call the object's destructor.

Errors: If Index isn't valid then Error ([353](#)) is called with CoIndexError.

See also: Delete ([349](#))

```
Program ex33;  
  
  { Program to demonstrate the TCollection.AtDelete method }  
  
Uses Objects,MyObject; { For TMyObject definition and registration }  
  
Var C : PCollection;  
    M : PMyObject;  
    I,InitMem : Longint;  
  
begin  
  Randomize;  
  C:=New(PCollection, Init(120,10));  
  InitMem:=Memavail;  
  Writeln ( 'Initial memory : ',InitMem);  
  For I:=1 to 100 do
```

```
begin
M:=New(PMyObject, Init);
M^.SetField(I-1);
C^.Insert(M);
end;
Writeln ('Added 100 Items. Memory available : ', Memavail);
Write ('Lost : ', Initmem-Memavail, ' bytes. ');
Write ('( Should be 100*', SizeOf(TMyObject));
Writeln ('=', 100*SizeOf(TMyObject), ') ');
With C^ do
  While Count>0 do AtDelete(Count-1);
Writeln ('Freed all objects. Memory available : ', Memavail);
Writeln ('Lost : ', Initmem-Memavail, ' bytes. ');
Dispose(C, Done);
end.
```

TCollection.ForEach

Declaration: Procedure TCollection.ForEach (Action: Pointer);

Description: ForEach calls Action for each element in the collection, and passes the element as an argument to Action.

Action is a procedural type variable that accepts a pointer as an argument.

Errors: None.

See also: FirstThat ([345](#)), LastThat ([345](#))

```
Program ex21;

{ Program to demonstrate the TCollection.ForEach method }

Uses Objects, MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer; P : PMyObject);

begin
  Writeln ('Field : ', P^.GetField);
end;

begin
  C:=New(PCollection, Init(100,10));
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(100-I);
      C^.Insert(M);
    end;
  Writeln ('Inserted ', C^.Count, ' objects ');
  C^.ForEach(@PrintField);
  C^.FreeAll;
  Dispose(C, Done);
end.
```

TCollection.SetLimit

Declaration: `Procedure TCollection.SetLimit (ALimit: Sw_Integer); Virtual;`

Description: `SetLimit` sets the maximum number of elements in the collection. `ALimit` must not be less than `Count`, and should not be larger than `MaxCollectionSize`

Errors: None.

See also: [Init \(342\)](#)

For an example, see [Pack \(346\)](#).

TCollection.Error

Declaration: `Procedure TCollection.Error (Code, Info: Integer); Virtual;`

Description: `Error` is called by the various `TCollection` methods in case of an error condition. The default behaviour is to make a call to `RunError` with an error of 212-Code.

This method can be overridden by descendent objects to implement a different error-handling.

Errors:

See also: [Abstract \(315\)](#)

TCollection.AtPut

Declaration: `Procedure TCollection.AtPut (Index: Sw_Integer; Item: Pointer);`

Description: `AtPut` sets the element at position `Index` in the collection to `Item`. Any previous value is overwritten.

Errors: If `Index` isn't valid then [Error \(353\)](#) is called with `CoIndexError`.

See also:

For an example, see [Pack \(346\)](#).

TCollection.AtInsert

Declaration: `Procedure TCollection.AtInsert (Index: Sw_Integer; Item: Pointer);`

Description: `AtInsert` inserts `Item` in the collection at position `Index`, shifting all elements by one position. In case the current limit is reached, the collection will try to expand with a call to `SetLimit`

Errors: If `Index` isn't valid then [Error \(353\)](#) is called with `CoIndexError`. If the collection fails to expand, then `coOverflow` is passed to `Error`.

See also: [Insert \(349\)](#)

Program ex34;

{ Program to demonstrate the TCollection.AtInsert method }

Uses Objects, MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;

```
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin
    Writeln ( 'Field : ',P^.GetField);
end;

begin
    Randomize;
    C:=New(PCollection,Init(120,10));
    Writeln ( 'Inserting 100 records at random places. ');
    For I:=1 to 100 do
        begin
            M:=New(PMyObject,Init);
            M^.SetField(I-1);
            If I=1 then
                C^.Insert(M)
            else
                With C^ do
                    AtInsert(Random(Count),M);
            end;
            Writeln ( 'Values : ');
            C^.Foreach( @PrintField );
            Dispose(C,Done);
        end.
```

TCollection.Store

Declaration: Procedure TCollection.Store (Var S: TStream);

Description: Store writes the collection to the stream S. It does this by writeing the current Count, Limit and Delta to the stream, and then writing each item to the stream.

The contents of the stream are then suitable for instantiating another collection with [Load \(342\)](#).

Errors: Errors returned are those by TStream.Put ([328](#)).

See also: [Load \(342\)](#), [PutItem \(354\)](#)

For an example, see `seepLoadTCollection.Load`.

TCollection.PutItem

Declaration: Procedure TCollection.PutItem (Var S: TStream; Item: Pointer); Virtual;

Description: PutItem writes Item to stream S. This method is used internaly by the TCollection object, and should not be called directly.

Errors: Errors are those returned by TStream.Put ([328](#)).

See also: [Store \(354\)](#), [GetItem \(344\)](#).

17.11 TSortedCollection

TSortedCollection is an abstract class, implementing a sorted collection. You should never use an instance of TSortedCollection directly, instead you should declare a descendent type, and override the **Compare** (357) method.

Because the collection is ordered, TSortedCollection overrides some TCollection methods, to provide faster routines for lookup.

The **Compare** (357) method decides how elements in the collection should be ordered. Since TCollection has no way of knowing how to order pointers, you must override the compare method.

Additionally, TCollection provides a means to filter out duplicates. if you set Duplicates to False (the default) then duplicates will not be allowed.

Here is the complete declaration of TSortedCollection

```
TYPE
  TSortedCollection = OBJECT (TCollection)
    Duplicates: Boolean; { Duplicates flag }
    Constructor Init (ALimit, ADelta: Sw_Integer);
    Constructor Load (Var S: TStream);
    Function KeyOf (Item: Pointer): Pointer; Virtual;
    Function IndexOf (Item: Pointer): Sw_Integer; Virtual;
    Function Compare (Key1, Key2: Pointer): Sw_Integer; Virtual;
    Function Search (Key: Pointer; Var Index: Sw_Integer): Boolean; Virtual;
    Procedure Insert (Item: Pointer); Virtual;
    Procedure Store (Var S: TStream);
  END;
  PSortedCollection = ^TSortedCollection;
```

In the subsequent examples, the following descendent of TSortedCollection is used:

```
Unit MySortC;
```

```
Interface
```

```
Uses Objects;
```

```
Type
```

```
  PMySortedCollection = ^TMySortedCollection;
  TMySortedCollection = Object(TSortedCollection)
    Function Compare (Key1,Key2 : Pointer): Sw_integer; virtual;
  end;
```

```
Implementation
```

```
Uses MyObject;
```

```
Function TMySortedCollection.Compare (Key1,Key2 : Pointer) :sw_integer;
```

```
begin
```

```
  Compare:=PMyobject(Key1)^.GetField - PMyObject(Key2)^.GetField;
end;
```

```
end.
```

TSortedCollection.Init

Declaration: Constructor `TSortedCollection.Init (ALimit, ADelta: Sw_Integer);`

Description: `Init` calls the inherited constructor (see `TCollection.Init` (342)) and sets the `Duplicates` flag to false.

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the inherited keyword.

Errors: None.

See also: `Load` (356), `Done` (343)

For an example, see

TSortedCollection.Load

Declaration: Constructor `Load (Var S: TStream);`

Description: `Load` calls the inherited constructor (see `TCollection.Load` (342)) and reads the `Duplicates` flag from the stream..

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the inherited keyword.

Errors: None.

See also: `Init` (356), `Done` (343)

For an example, see `TCollection.Load` (342).

TSortedCollection.KeyOf

Declaration: Function `TSortedCollection.KeyOf (Item: Pointer): Pointer; Virtual;`

Description: `KeyOf` returns the key associated with `Item`. `TSortedCollection` returns the item itself as the key, descendent objects can override this method to calculate a (unique) key based on the item passed (such as hash values).

Keys are used to sort the objects, they are used to search and sort the items in the collection. If descendent types override this method then it allows possibly for faster search/sort methods based on keys rather than on the objects themselves.

Errors: None.

See also: `IndexOf` (356), `Compare` (357).

TSortedCollection.IndexOf

Declaration: Function `TSortedCollection.IndexOf (Item: Pointer): Sw_Integer; Virtual;`

Description: `IndexOf` returns the index of `Item` in the collection. It searches for the object based on it's key. If duplicates are allowed, then it returns the index of last object that matches `Item`.

In case `Item` is not found in the collection, -1 is returned.

Errors: None.

See also: `Search` (357), `Compare` (357).

For an example, see `TCollection.IndexOf` (344)

TSortedCollection.Compare

Declaration: `Function TSortedCollection.Compare (Key1, Key2: Pointer): Sw_Integer; Virtual;`

Description: Compare is an abstract method that should be overridden by descendent objects in order to compare two items in the collection. This method is used in the [Search \(357\)](#) method and in the [Insert \(358\)](#) method to determine the ordering of the objects.

The function should compare the two keys of items and return the following function results:

Result < 0 If Key1 is logically before Key2 (Key1<Key2)

Result = 0 If Key1 and Key2 are equal. (Key1=Key2)

Result > 0 If Key1 is logically after Key2 (Key1>Key2)

Errors: An 'abstract run-time error' will be generated if you call `TSortedCollection.Compare` directly.

See also: [IndexOf \(356\)](#), [Search \(357\)](#)

Unit MySortC;

Interface

Uses Objects;

Type

PMysortedCollection = ^TMySortedCollection;

TMySortedCollection = **Object**(TSortedCollection)

Function Compare (Key1,Key2 : Pointer) : Sw_integer; **virtual**;
end;

Implementation

Uses MyObject;

Function TMySortedCollection.Compare (Key1,Key2 : Pointer) : sw_integer;

begin

Compare:=PMYobject(Key1)^.GetField - PMYObject(Key2)^.GetField;

end;

end.

TSortedCollection.Search

Declaration: `Function TSortedCollection.Search (Key: Pointer; Var Index: Sw_Integer): Boolean;Virtual;`

Description: Search looks for the item with key Key and returns the position of the item (if present) in the collection in Index.

Instead of a linear search as TCollection does, TSortedCollection uses a binary search based on the keys of the objects. It uses the [Compare \(357\)](#) function to implement this search.

If the item is found, Search returns True, otherwise False is returned.

Errors: None.

See also: [IndexOf \(344\)](#).

```
Program ex36;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects, MyObject, MySortC;
{ For TMyObject, TMySortedCollection definition and registration }

Var C : PSortedCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer; P : PMyObject);

begin
    WriteLn ( 'Field : ', P^.GetField );
end;

begin
    Randomize;
    C:=New(PMySortedCollection, Init(120,10));
    C^.Duplicates:=True;
    WriteLn ( 'Inserting 100 records at random places.' );
    For I:=1 to 100 do
        begin
            M:=New(PMyObject, Init);
            M^.SetField(Random(100));
            C^.Insert(M)
        end;
    M:=New(PMyObject, Init);
    Repeat;
        Write ( 'Value to search for (-1 stops) :' );
        read ( I );
        If I<>-1 then
            begin
                M^.SetField(I);
                If Not C^.Search (M, I) then
                    WriteLn ( 'No such value found' )
                else
                    begin
                        Write ( 'Value ', PMyObject(C^.At(I))^ .GetField );
                        WriteLn ( ' present at position ', I );
                    end;
            end;
        Until I=-1;
        Dispose(M, Done);
        Dispose(C, Done);
    end.
```

TSortedCollection.Insert

Declaration: `Procedure TSortedCollection.Insert (Item: Pointer); Virtual;`

Description: `Insert` inserts an item in the collection at the correct position, such that the collection is ordered at all times. You should never use `Atinsert` ([353](#)), since then the collection ordering is not guaranteed.

If `Item` is already present in the collection, and `Duplicates` is `False`, the item will not be inserted.

Errors: None.

See also: [AtInsert \(353\)](#)

```
Program ex35;  
  
  { Program to demonstrate the TSortedCollection.Insert method }  
  
Uses Objects, MyObject, MySortC;  
  { For TMyObject, TMySortedCollection definition and registration }  
  
Var C : PSortedCollection;  
      M : PMyObject;  
      I : Longint;  
  
Procedure PrintField (Dummy: Pointer; P : PMyObject);  
  
  begin  
    WriteLn ( 'Field : ', P^.GetField );  
  end;  
  
  begin  
    Randomize;  
    C:=New( PMySortedCollection, Init(120,10));  
    WriteLn ( 'Inserting 100 records at random places.' );  
    For I:=1 to 100 do  
      begin  
        M:=New(PMyObject, Init);  
        M^.SetField(Random(100));  
        C^.Insert(M)  
      end;  
    WriteLn ( 'Values : ' );  
    C^.Foreach( @PrintField );  
    Dispose(C, Done);  
  end.
```

TSortedCollection.Store

Declaration: `Procedure TSortedCollection.Store (Var S: TStream);`

Description: `Store` writes the collection to the stream `S`. It does this by calling the inherited `TCollection.Store` ([354](#)), and then writing the `Duplicates` flag to the stream.

After a `Store`, the collection can be loaded from the stream with the constructor `Load` ([356](#))

Errors: Errors can be those of `TStream.Put` ([328](#)).

See also: `Load` ([356](#))

For an example, see `TCollection.Load` ([342](#)).

17.12 TStringCollection

The TStringCollection object manages a sorted collection of pascal strings. To this end, it overrides the Compare (357) method of TSortedCollection, and it introduces methods to read/write strings from a stream.

Here is the full declaration of the TStringCollection object:

```
TYPE
  TStringCollection = OBJECT (TSortedCollection)
    Function GetItem (Var S: TStream): Pointer; Virtual;
    Function Compare (Key1, Key2: Pointer): Sw_Integer; Virtual;
    Procedure FreeItem (Item: Pointer); Virtual;
    Procedure PutItem (Var S: TStream; Item: Pointer); Virtual;
  END;
  PStringCollection = ^TStringCollection;
```

TStringCollection.GetItem

Declaration: Function TStringCollection.GetItem (Var S: TStream): Pointer; Virtual;

Description: GetItem reads a string from the stream S and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of TStream.ReadStr (326).

See also: PutItem (361)

TStringCollection.Compare

Declaration: Function TStringCollection.Compare (Key1, Key2: Pointer): Sw_Integer; Virtual;

Description: TStringCollection overrides the Compare function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns the following results:

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare (357)

Program ex37;

{ Program to demonstrate the TStringCollection.Compare method }

Uses Objects;

```
Var C : PStringCollection;
    S : String;
    I : longint;
```

```
begin
  Randomize;
  C:=New(PStringCollection, Init(120,10));
  C^.Duplicates:=True; { Duplicates allowed }
  Writeln ('Inserting 100 records at random places. ');
  For I:=1 to 100 do
    begin
      Str(Random(100),S);
      S:='String with value '+S;
      C^.Insert(NewStr(S));
    end;
  For I:=0 to 98 do
    With C^ do
      If Compare (At(i),At(I+1))=0 then
        Writeln ('Duplicate string found at position ',i);
  Dispose(C,Done);
end.
```

TStringCollection.FreeItem

Declaration: Procedure TStringCollection.FreeItem (Item: Pointer); Virtual;

Description: TStringCollection overrides FreeItem so that the string pointed to by Item is disposed from memory.

Errors: None.

See also: TCollection.FreeItem ([351](#))

TStringCollection.PutItem

Declaration: Procedure TStringCollection.PutItem (Var S: TStream; Item: Pointer); Virtual;

Description: PutItem writes the string pointed to by Item to the stream S.

This method is primarily used in the Load and Store methods, and should not be used directly.

Errors: Errors are those of TStream.WriteString ([329](#)).

See also: GetItem ([360](#))

17.13 TStrCollection

The TStrCollection object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the Compare ([357](#)) method of TSortedCollection, and it introduces methods to read/write strings from a stream.

Here is the full declaration of the TStrCollection object:

```
TYPE
  TStrCollection = OBJECT (TSortedCollection)
    Function Compare (Key1, Key2: Pointer): Sw_Integer; Virtual;
    Function GetItem (Var S: TStream): Pointer; Virtual;
    Procedure FreeItem (Item: Pointer); Virtual;
```

```
Procedure PutItem (Var S: TStream; Item: Pointer); Virtual;  
END;  
PStrCollection = ^TStrCollection;
```

TStrCollection.GetItem

Declaration: Function TStrCollection.GetItem (Var S: TStream): Pointer; Virtual;

Description: GetItem reads a null-terminated string from the stream S and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of TStream.StrRead ([325](#)).

See also: PutItem ([363](#))

TStrCollection.Compare

Declaration: Function TStrCollection.Compare (Key1, Key2: Pointer): Sw_Integer;
Virtual;

Description: TStrCollection overrides the Compare function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare ([357](#))

```
Program ex38;  
  
{ Program to demonstrate the TStrCollection.Compare method }  
  
Uses Objects, Strings;  
  
Var C : PStrCollection;  
      S : String;  
      I : longint;  
      P : Pchar;  
  
begin  
  Randomize;  
  C:=New(PStrCollection, Init(120,10));  
  C^.Duplicates:=True; { Duplicates allowed }  
  Writeln ('Inserting 100 records at random places. ');  
  For I:=1 to 100 do  
    begin  
      Str(Random(100),S);  
      S:='String with value '+S;  
      P:=StrAlloc(Length(S)+1);  
      C^.Insert(StrPCopy(P,S));  
    end;  
  For I:=0 to 98 do
```

```
With C^ do
  If Compare (At(l), At(l+1))=0 then
    WriteLn ('Duplicate string found at position ', l);
  Dispose(C, Done);
end.
```

TStrCollection.FreeItem

Declaration: Procedure TStrCollection.FreeItem (Item: Pointer); Virtual;

Description: TStrCollection overrides FreeItem so that the string pointed to by Item is disposed from memory.

Errors: None.

See also: TCollection.FreeItem ([351](#))

TStrCollection.PutItem

Declaration: Procedure TStrCollection.PutItem (Var S: TStream; Item: Pointer); Virtual;

Description: PutItem writes the string pointed to by Item to the stream S.

This method is primarily used in the Load and Store methods, and should not be used directly.

Errors: Errors are those of TStream.StrWrite ([329](#)).

See also: GetItem ([362](#))

17.14 TUnSortedStrCollection

The TUnSortedStrCollection object manages an unsorted list of objects. To this end, it overrides the TSortedCollection.Insert ([358](#)) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the Search ([357](#)) and IndexOf ([344](#)) methods will not work on an unsorted string collection.

Here is the full declaration of the TUnsortedStrCollection object:

```
TYPE
  TUnSortedStrCollection = OBJECT (TStringCollection)
    Procedure Insert (Item: Pointer); Virtual;
  END;
  PUnSortedStrCollection = ^TUnSortedStrCollection;
```

TUnSortedStrCollection.Insert

Declaration: Procedure TUnSortedStrCollection.Insert (Item: Pointer); Virtual;

Description: Insert inserts a string at the end of the collection, instead of on its alphabetical place, resulting in an unsorted collection of strings.

Errors:

See also:

```
Program ex39;

{ Program to demonstrate the TUnsortedStrCollection.Insert method }

Uses Objects, Strings;

Var C : PUnsortedStrCollection;
    S : String;
    I : longint;
    P : Pchar;

begin
  Randomize;
  C:=New(PUnsortedStrCollection, Init(120,10));
  WriteLn ( 'Inserting 100 records at random places.' );
  For I:=1 to 100 do
    begin
      Str(Random(100),S);
      S:='String with value '+S;
      P:=StrAlloc(Length(S)+1);
      C^.Insert(StrPCopy(P,S));
    end;
  For I:=0 to 99 do
    WriteLn ( I:2, ': ', PChar(C^.At(i)));
  Dispose(C,Done);
end.
```

17.15 TResourceCollection

A TResourceCollection manages a collection of resource names. It stores the position and the size of a resource, as well as the name of the resource. It stores these items in records that look like this:

```
TYPE
  TResourceItem = packed RECORD
    Posn: LongInt;
    Size: LongInt;
    Key : String;
  End;
  PResourceItem = ^TResourceItem;
```

It overrides some methods of TStringCollection in order to accomplish this.

Remark that the TResourceCollection manages the names of the resources and their associated positions and sizes, it doesn't manage the resources themselves.

Here is the full declaration of the TResourceCollection object:

```
TYPE
  TResourceCollection = OBJECT (TStringCollection)
    Function KeyOf (Item: Pointer): Pointer; Virtual;
    Function GetItem (Var S: TStream): Pointer; Virtual;
    Procedure FreeItem (Item: Pointer); Virtual;
    Procedure PutItem (Var S: TStream; Item: Pointer); Virtual;
END;
```

```
PResourceCollection = ^TResourceCollection;
```

TResourceCollection.KeyOf

Declaration: `Function TResourceCollection.KeyOf (Item: Pointer): Pointer; Virtual;`

Description: `KeyOf` returns the key of an item in the collection. For resources, the key is a pointer to the string with the resource name.

Errors: None.

See also: `TStringCollection.Compare` ([360](#))

TResourceCollection.GetItem

Declaration: `Function TResourceCollection.GetItem (Var S: TStream): Pointer; Virtual;`

Description: `GetItem` reads a resource item from the stream `S`. It reads the position, size and name from the stream, in that order. It DOES NOT read the resource itself from the stream.

The resulting item is not inserted in the collection. This call is mainly for internal use by the `TCollection.Load` ([342](#)) method.

Errors: Errors returned are those by `TStream.Read` ([330](#))

See also: `TCollection.Load` ([342](#)), `TStream.Read` ([330](#))

TResourceCollection.FreeItem

Declaration: `Procedure TResourceCollection.FreeItem (Item: Pointer); Virtual;`

Description: `FreeItem` releases the memory occupied by `Item`. It de-allocates the name, and then the resource item record.

It does NOT remove the item from the collection.

Errors: None.

See also: `TCollection.FreeItem` ([351](#))

TResourceCollection.PutItem

Declaration: `Procedure TResourceCollection.PutItem (Var S: TStream; Item: Pointer); Virtual;`

Description: `PutItem` writes `Item` to the stream `S`. It does this by writing the position and size and name of the resource item to the stream.

This method is used primarily by the `Store` ([354](#)) method.

Errors: Errors returned are those by `TStream.Write` ([330](#)).

See also: `Store` ([354](#))

17.16 TResourceFile

TYPE

```
TResourceFile = OBJECT (TObject)
    Stream : PStream; { File as a stream }
    Modified: Boolean; { Modified flag }
    Constructor Init (AStream: PStream);
    Destructor Done; Virtual;
    Function Count: Sw_Integer;
    Function KeyAt (I: Sw_Integer): String;
    Function Get (Key: String): PObject;
    Function SwitchTo (AStream: PStream; Pack: Boolean): PStream;
    Procedure Flush;
    Procedure Delete (Key: String);
    Procedure Put (Item: PObject; Key: String);
END;
PResourceFile = ^TResourceFile;
```

TResourceFile Fields

TResourceFile has the following fields:

Stream contains the (file) stream that has the executable image and the resources. It can be initialized by the [Init \(366\)](#) constructor call.

Modified is set to True if one of the resources has been changed. It is set by the [SwitchTo \(366\)](#), [Delete \(368\)](#) and [Put \(368\)](#) methods. Calling [Flush \(367\)](#) will clear the Modified flag.

TResourceFile.Init

Declaration: `Constructor TResourceFile.Init (AStream: PStream);`

Description: Init instantiates a new instance of a TResourceFile object. If AStream is not nil then it is considered as a stream describing an executable image on disk.

Init will try to position the stream on the start of the resources section, and read all resources from the stream.

Errors: None.

See also: [Done \(366\)](#)

TResourceFile.Done

Declaration: `Destructor TResourceFile.Done; Virtual;`

Description: Done cleans up the instance of the TResourceFile Object. If Stream was specified at initialization, then Stream is disposed of too.

Errors: None.

See also: [Init \(366\)](#)

TResourceFile.Count

Declaration: `Function TResourceFile.Count: Sw_Integer;`

Description: `Count` returns the number of resources. If no resources were read, zero is returned.

Errors: None.

See also: `Init` ([366](#))

TResourceFile.KeyAt

Declaration: `Function TResourceFile.KeyAt (I: Sw_Integer): String;`

Description: `KeyAt` returns the key (the name) of the `I`-th resource.

Errors: In case `I` is invalid, `TCollection.Error` will be executed.

See also: `Get` ([367](#))

TResourceFile.Get

Declaration: `Function TResourceFile.Get (Key: String): PObject;`

Description: `Get` returns a pointer to a instance of a resource identified by `Key`. If `Key` cannot be found in the list of resources, then `Nil` is returned.

Errors: Errors returned may be those by `TStream.Get`

See also:

TResourceFile.SwitchTo

Declaration: `Function TResourceFile.SwitchTo (AStream: PStream; Pack: Boolean): PStream;`

Description: `SwitchTo` switches to a new stream to hold the resources in. `AStream` will be the new stream after the call to `SwitchTo`.

If `Pack` is true, then all the known resources will be copied from the current stream to the new stream (`AStream`). If `Pack` is False, then only the current resource is copied.

The return value is the value of the original stream: `Stream`.

The `Modified` flag is set as a consequence of this call.

Errors: Errors returned can be those of `TStream.Read` ([330](#)) and `TStream.Write` ([330](#)).

See also: `Flush` ([367](#))

TResourceFile.Flush

Declaration: `Procedure TResourceFile.Flush;`

Description: If the `Modified` flag is set to True, then `Flush` writes the resources to the stream `Stream`. It sets the `Modified` flag to true after that.

Errors: Errors can be those by `TStream.Seek` ([329](#)) and `TStream.Write` ([330](#)).

See also: `SwitchTo` ([367](#))

TResourceFile.Delete

Declaration: `Procedure TResourceFile.Delete (Key: String);`

Description: Delete deletes the resource identified by Key from the collection. It sets the Modified flag to true.

Errors: None.

See also: Flush ([367](#))

TResourceFile.Put

Declaration: `Procedure TResourceFile.Put (Item: PObject; Key: String);`

Description: Put sets the resource identified by Key to Item. If no such resource exists, a new one is created. The item is written to the stream.

Errors: Errors returned may be those by TStream.Put ([328](#)) and TStream.Seek

See also: Get ([367](#))

17.17 TStringList

A TStringList object can be used to read a collection of strings stored in a stream. If you register this object with the RegisterType ([315](#)) function, you cannot register the TStrListMaker object.

This is the public declaration of the TStringList object:

```
TYPE
  TStrIndexRec = Packed RECORD
    Key, Count, Offset: Word;
  END;

  TStrIndex = Array [0..9999] Of TStrIndexRec;
  PStrIndex = ^TStrIndex;

  TStringList = OBJECT (TObject)
    Constructor Load (Var S: TStream);
    Destructor Done; Virtual;
    Function Get (Key: Sw_Word): String;
  END;
  PStringList = ^TStringList;
```

TStringList.Load

Declaration: `Constructor TStringList.Load (Var S: TStream);`

Description: The Load constructor reads the TStringList object from the stream S. It also reads the descriptions of the strings from the stream. The string descriptions are stored as an array of TStrIndexrec records, where each record describes a string on the stream. These records are kept in memory.

Errors: If an error occurs, a stream error is triggered.

See also: Done ([369](#))

TStringList.Done

Declaration: Destructor `TstringList.Done; Virtual;`

Description: The Done destructor frees the memory occupied by the string descriptions, and destroys the object.

Errors: None.

See also: Load ([368](#)), TObject.Done ([323](#))

TStringList.Get

Declaration: Function `TStringList.Get (Key: Sw_Word): String;`

Description: Get reads the string with key Key from the list of strings on the stream, and returns this string. If there is no string with such a key, an empty string is returned.

Errors: If no string with key Key is found, an empty string is returned. A stream error may result if the stream doesn't contain the needed strings.

See also: TStrListMaker.Put ([370](#))

17.18 TStrListMaker

The TStrListMaker object can be used to generate a stream with strings, which can be read with the TStringList object. If you register this object with the RegisterType ([315](#)) function, you cannot register the TStringList object.

This is the public declaration of the TStrListMaker object:

```
TYPE
  TStrListMaker = OBJECT (TObject)
    Constructor Init (AStrSize, AIndexSize: Sw_Word);
    Destructor Done; Virtual;
    Procedure Put (Key: SwWord; S: String);
    Procedure Store (Var S: TStream);
  END;
  PStrListMaker = ^TStrListMaker;
```

TStrListMaker.Init

Declaration: Constructor `TStrListMaker.Init (AStrSize, AIndexSize: SwWord);`

Description: The Init constructor creates a new instance of the TstrListMaker object. It allocates AStrSize bytes on the heap to hold all the strings you wish to store. It also allocates enough room for AIndexSize key description entries (of the type TStrIndexrec).

AStrSize must be large enough to contain all the strings you wish to store. If not enough memory is allocated, other memory will be overwritten. The same is true for AIndexSize : maximally AIndexSize strings can be written to the stream.

Errors: None.

See also: TObject.Init ([322](#)), Done ([370](#))

TStrListMaker.Done

Declaration: `Destructor TStrListMaker.Done; Virtual;`

Description: The `Done` destructor de-allocates the memory for the index description records and the string data, and then destroys the object.

Errors: None.

See also: `TObject.Done` ([323](#)), `Init` ([369](#))

TStrListMaker.Put

Declaration: `Procedure TStrListMaker.Put (Key: Sw_Word; S: String);`

Description: `Put` adds the string `S` with key `Key` to the collection of strings. This action doesn't write the string to a stream. To write the strings to the stream, see the `Store` ([370](#)) method.

Errors: None.

See also: `Store` ([370](#)).

TStrListMaker.Store

Declaration: `Procedure TStrListMaker.Store (Var S: TStream);`

Description: `Store` writes the collection of strings to the stream `S`. The collection can then be read with the `TStringList` object.

Errors: A stream error may occur when writing the strings to the stream.

See also: `TStringList.Load` ([368](#)), `Put` ([370](#)).

Chapter 18

The PORTS unit

18.1 Introduction

The ports unit implements the `port` constructs found in Turbo Pascal. It uses classes and default array properties to do this.

The unit exists on LINUX, OS/2 and DOS. It is implemented only for compatibility with Turbo Pascal. It's usage is discouraged, because using ports is not portable programming, and the operating system may not even allow it (for instance WINDOWS).

Under LINUX, your program must be run as root, or the `IOPerm` call must be set in order to set appropriate permissions on the port access.

18.2 Types, constants and variables

Types

The following types are defined to implement the port access.

```
tport = class
  protected
    procedure writeport(p : longint; data : byte);
    function readport(p : longint) : byte;
  public
    property pp[w : longint] : byte read readport write writeport; default;
end;
```

```
tportw = class
  protected
    procedure writeport(p : longint; data : word);
    function readport(p : longint) : word;
  public
    property pp[w : longint] : word read readport write writeport; default;
end;
```

```
tportl = class
  Protected
    procedure writeport(p : longint; data : longint);
    function readport(p : longint) : longint;
```

```
Public
  property pp[w : Longint] : longint read readport write writeport;default;
end;
```

Each of these types allows access to the ports using respectively, a byte, a word or a longint sized argument.

Since there is a default property for each of this types, a sentence as

```
port[221]:=12;
```

Will result in the byte 12 being written to port 221, if port is defined as a variable of type tport

variables

The following variables are defined:

```
port,
portb : tport;
portw : tportw;
portl : tportl;
```

They allow access to the ports in a Turbo Pascal compatible way.

Chapter 19

The PRINTER unit.

This chapter describes the PRINTER unit for Free Pascal. It was written for DOS by Florian klämpfl, and it was written for LINUX by Michaël Van Canneyt, and has been ported to WINDOWS as well. Its basic functionality is the same for both systems, although there are minor differences on LINUX.

The chapter is divided in 2 sections:

- The first section lists types, constants and variables from the interface part of the unit.
- The second section describes the functions defined in the unit.

19.1 Types, Constants and variables :

```
var
  Lst : text;
```

Lst is the standard printing device.

On LINUX, Lst is set up using AssignLst (' /tmp/PID.lst '). You can change this behaviour at compile time, setting the DefFile constant.

19.2 Procedures and functions

AssignLst

Declaration: Procedure AssignLst (Var F : text; ToFile : string[255]);

Description: LINUX only.

Assigns to F a printing device. ToFile is a string with the following form:

- ' |filename options' : This sets up a pipe with the program filename, with the given options, such as in the popen() call.
- ' filename' : Prints to file filename. Filename can contain the string 'PID' (No Quotes), which will be replaced by the PID of your program. When closing lst, the file will be sent to lpr and deleted. (lpr should be in PATH)
- ' filename | ' Idem as previous, only the file is NOT sent to lpr, nor is it deleted. (useful for opening /dev/printer or for later printing)

Errors: Errors are reported in Linuxerror.

See also: `lpr` (1)

```
program testprn;

uses printer;

var i : integer;
    f : text;

begin
  writeln ('Test of printer unit');
  writeln ('Writing to lst...');
  for i:=1 to 80 do writeln (lst,'This is line ',i,'.'#13);
  close (lst);
  writeln ('Done. ');
  {$ifdef linux}
  writeln ('Writing to pipe...');
  assignlst (f,'|/usr/bin/lpr -m');
  rewrite (f);
  for i:=1 to 80 do writeln (f,'This is line ',i,'.'#13);
  close (f);
  writeln ('Done. ')
  {$endif}
end.
```

Chapter 20

The SOCKETS unit.

This chapter describes the SOCKETS unit for Free Pascal. it was written for LINUX by Michaël Van Canneyt, and ported to WINDOWS by Florian Klaempfl. The chapter is divided in 2 sections:

- The first section lists types, constants and variables from the interface part of the unit.
- The second section describes the functions defined in the unit.

20.1 Types, Constants and variables :

The following constants identify the different socket types, as needed in the `Socket` ([385](#)) call.

```
SOCK_STREAM      = 1; { stream (connection) socket    }
SOCK_DGRAM       = 2; { datagram (conn. less) socket   }
SOCK_RAW         = 3; { raw socket                     }
SOCK_RDM         = 4; { reliably-delivered message    }
SOCK_SEQPACKET   = 5; { sequential packet socket      }
SOCK_PACKET      = 10;
```

The following constants determine the socket domain, they are used in the `Socket` ([385](#)) call.

```
AF_UNSPEC        = 0;
AF_UNIX          = 1; { Unix domain sockets           }
AF_INET          = 2; { Internet IP Protocol          }
AF_AX25          = 3; { Amateur Radio AX.25           }
AF_IPX           = 4; { Novell IPX                    }
AF_APPLETALK     = 5; { Appletalk DDP                 }
AF_NETROM        = 6; { Amateur radio NetROM          }
AF_BRIDGE        = 7; { Multiprotocol bridge          }
AF_AAL5          = 8; { Reserved for Werner's ATM     }
AF_X25           = 9; { Reserved for X.25 project     }
AF_INET6         = 10; { IP version 6                 }
AF_MAX           = 12;
```

The following constants determine the protocol family, they are used in the `Socket` ([385](#)) call.

```
PF_UNSPEC        = AF_UNSPEC;
PF_UNIX          = AF_UNIX;
```

```
PF_INET      = AF_INET;
PF_AX25      = AF_AX25;
PF_IPX       = AF_IPX;
PF_APPLETALK = AF_APPLETALK;
PF_NETROM    = AF_NETROM;
PF_BRIDGE    = AF_BRIDGE;
PF_AAL5      = AF_AAL5;
PF_X25       = AF_X25;
PF_INET6     = AF_INET6;
PF_MAX       = AF_MAX;
```

The following types are used to store different kinds of addresses for the [Bind \(378\)](#), [Recv \(383\)](#) and [Send \(383\)](#) calls.

```
TSockAddr = packed Record
  family:word;
  data :array [0..13] of char;
end;
TUnixSockAddr = packed Record
  family:word;
  path:array[0..108] of char;
end;
TInetSockAddr = packed Record
  family:Word;
  port :Word;
  addr :Cardinal;
  pad :array [1..8] of byte;
end;
```

The following type is returned by the [SocketPair \(385\)](#) call.

```
TSockArray = Array[1..2] of Longint;
```

20.2 Functions and Procedures

Accept

Declaration: `Function Accept (Sock:Longint;Var Addr;Var Addrlen:Longint) : Longint;`

Description: `Accept` accepts a connection from a socket `Sock`, which was listening for a connection. If a connection is accepted, a file descriptor is returned. On error `-1` is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The `Accept` call fills the address of the connecting entity in `Addr`, and sets its length in `Addrlen`. `Addr` should be pointing to enough space, and `Addrlen` should be set to the amount of space available, prior to the call.

Errors: On error, `-1` is returned, and errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EOPNOTSUPPThe socket type doesn't support the `Listen` operation.

SYS_EFAULT`Addr` points outside your address space.

SYS_EWOULDBLOCKThe requested operation would block the process.

See also: Listen ([382](#)), Connect ([379](#))

```
Program server;

{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Server Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}

uses Linux, Sockets;
const
  SPath='ServerSoc';

Var
  FromName : string;
  Buffer    : string[255];
  S        : Longint;
  Sin, Sout : Text;

procedure perror (const S:string);
begin
  writeln (S, SocketError);
  halt(100);
end;

begin
  S:=Socket (AF_UNIX, SOCK_STREAM, 0);
  if SocketError<>0 then
    Perror ('Server : Socket : ');
  Unlink(SPath);
  if not Bind(S, SPath) then
    Perror ('Server : Bind : ');
  if not Listen (S, 1) then
    Perror ('Server : Listen : ');
  Writeln('Waiting for Connect from Client, run now sock_cli in an other tty');
  if not Accept (S, FromName, Sin, Sout) then
    Perror ('Server : Accept : '+fromname);
  Reset(Sin);
  ReWrite(Sout);
  Writeln(Sout, 'Message From Server');
  Flush(Sout);
  while not eof(sin) do
    begin
      Readln(Sin, Buffer);
      Writeln('Server : read : ', buffer);
    end;
  Unlink(SPath);
end.
```

Accept

Declaration: Function Accept (Sock:longint;var addr:string;var SockIn,SockOut:text)
: Boolean;

Description: This is an alternate form of the [Accept \(376\)](#) command. It is equivalent to subsequently calling the regular [Accept \(376\)](#) function and the [Sock2Text \(385\)](#) function. The function returns `True` if successful, `False` otherwise.

Errors: The errors are those of [Accept \(376\)](#).

See also: [Accept \(376\)](#)

Accept

Declaration: `Function Accept (Sock:longint;var addr:string;var SockIn,SockOut:File) : Boolean;`

Description: This is an alternate form of the [Accept \(376\)](#) command. It is equivalent to subsequently calling the regular [Accept \(376\)](#) function and the [Sock2File \(384\)](#) function. The `Addr` parameter contains the name of the unix socket file to be opened. The function returns `True` if successful, `False` otherwise.

Errors: The errors are those of [Accept \(376\)](#).

See also: [Accept \(376\)](#)

Accept

Declaration: `Function Accept (Sock:longint;var addr:TInetSockAddr;var SockIn,SockOut:File) : Boolean;`

Description: This is an alternate form of the [Accept \(376\)](#) command. It is equivalent to subsequently calling the regular [Accept \(376\)](#) function and the [Sock2File \(384\)](#) function. The `Addr` parameter contains the parameters of the internet socket that should be opened. The function returns `True` if successful, `False` otherwise.

Errors: The errors are those of [Accept \(376\)](#).

See also: [Accept \(376\)](#)

Bind

Declaration: `Function Bind (Sock:Longint;Var Addr;AddrLen:Longint) : Boolean;`

Description: `Bind` binds the socket `Sock` to address `Addr`. `Addr` has length `AddrLen`. The function returns `True` if the call was successful, `False` if not.

Errors: Errors are returned in `SocketError` and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_EINVALThe socket is already bound to an address,

SYS_EACCESSAddress is protected and you don't have permission to open it.

More errors can be found in the Unix man pages.

See also: [Socket \(385\)](#)

Bind

Declaration: `Function Bind (Sock:longint;const addr:string) : boolean;`

Description: This is an alternate form of the Bind command. This form of the Bind command is equivalent to subsequently calling `Str2UnixSockAddr` (385) and the regular `Bind` (378) function. The function returns `True` if successful, `False` otherwise.

Errors: Errors are those of the regular `Bind` (378) command.

See also: `Bind` (378)

Connect

Declaration: `Function Connect (Sock:Longint;Var Addr;Addrlen:Longint) : Longint;`

Description: `Connect` opens a connection to a peer, whose address is described by `Addr`. `AddrLen` contains the length of the address. The type of `Addr` depends on the kind of connection you're trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The `Connect` function returns a file descriptor if the call was successful, `-1` in case of error.

Errors: On error, `-1` is returned and errors are reported in `SocketError`.

See also: `Listen` (382), `Bind` (378), `Accept` (376)

Program Client;

```
{  
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman  
  Client Version, First Run sock_svr to let it create a socket and then  
  sock_cli to connect to that socket  
}
```

uses Sockets, Linux;

```
procedure PError(const S : string);  
begin  
  writeln(S, SocketError);  
  halt(100);  
end;
```

Var

```
Saddr   : String[25];  
Buffer  : string [255];  
S        : Longint;  
Sin, Sout : Text;  
i         : integer;
```

begin

```
S:=Socket (AF_UNIX,SOCK_STREAM,0);  
if SocketError<>0 then  
  PError('Client : Socket : ');  
Saddr:='ServerSoc';  
if not Connect (S,Saddr,Sin,Sout) then  
  PError('Client : Connect : ');  
Reset(Sin);  
ReWrite(Sout);  
Buffer:='This is a textstring sent by the Client.';
```

```
    for i:=1 to 10 do
        Writeln(Sout, Buffer);
    Flush(Sout);
    Readln(SIn, Buffer);
    Writeln(Buffer);
    Close(sout);
end.
```

Connect

Declaration: Function Connect (Sock:longint;const addr:string;var SockIn,SockOut:text)
: Boolean;

Description: This is an alternate form of the [Connect \(379\)](#) command. It is equivalent to subsequently calling the regular [Connect \(379\)](#) function and the [Sock2Text \(385\)](#) function. The function returns True if successful, False otherwise.

Errors: The errors are those of [Connect \(379\)](#).

See also: [Connect \(379\)](#)

Connect

Declaration: Function Connect (Sock:longint;const addr:string;var SockIn,SockOut:file)
: Boolean;

Description: This is an alternate form of the [Connect \(379\)](#) command. The parameter `addr` contains the name of the unix socket file to be opened. It is equivalent to subsequently calling the regular [Connect \(379\)](#) function and the [Sock2File \(384\)](#) function. The function returns True if successful, False otherwise.

Errors: The errors are those of [Connect \(379\)](#).

See also: [Connect \(379\)](#)

Connect

Declaration: Function Connect (Sock:longint;const addr: TInetSockAddr;var SockIn,SockOut:file)
: Boolean;

Description: This is another alternate form of the [Connect \(379\)](#) command. It is equivalent to subsequently calling the regular [Connect \(379\)](#) function and the [Sock2File \(384\)](#) function. The `Addr` parameter contains the parameters of the internet socket to connect to. The function returns True if successful, False otherwise.

Errors: The errors are those of [Connect \(379\)](#).

See also: [Connect \(379\)](#)

```
program pfinger;

uses sockets, errors;

Var Addr : TInetSockAddr;
    S : Longint;
    Sin, Sout : Text;
```

```
Line : string;

begin
  Addr.family:=AF_INET;
  { port 79 in network order }
  Addr.port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.addr:=((1 shl 24) or 127);
  S:=Socket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,ADDR,SIN,SOUT) Then
    begin
      Writeln ('Couldn't connect to localhost');
      Writeln ('Socket error : ',strerror(SocketError));
      halt(1);
    end;
  rewrite (sout);
  reset(sin);
  writeln (sout,paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
      readln (Sin,line);
      writeln (line);
    end;
  Shutdown(s,2);
  close (sin);
  close (sout);
end.
```

GetPeerName

Declaration: Function GetPeerName (Sock:Longint;Var Addr;Var Addrlen:Longint) : Longint;

Description: GetPeerName returns the name of the entity connected to the specified socket Sock. The Socket must be connected for this call to work. Addr should point to enough space to store the name, the amount of space pointed to should be set in Addrlen. When the function returns successfully, Addr will be filled with the name, and Addrlen will be set to the length of Addr.

Errors: Errors are reported in SocketError, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOBUFSThe system doesn't have enough buffers to perform the operation.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULTAddr points outside your address space.

SYS_ENOTCONNThe socket isn't connected.

See also: Connect ([379](#)), Socket ([385](#)), connect (2)

GetSocketName

Declaration: Function GetSocketName (Sock:Longint;Var Addr;Var Addrlen:Longint) : Longint;

Description: GetSockName returns the current name of the specified socket Sock. Addr should point to enough space to store the name, the amount of space pointed to should be set in Addrlen. When

the function returns successfully, `Addr` will be filled with the name, and `AddrLen` will be set to the length of `Addr`.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOBUFSThe system doesn't have enough buffers to perform the operation.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULT`Addr` points outside your address space.

See also: `Bind` ([378](#))

GetSocketOptions

Declaration: `Function GetSocketOptions (Sock,Level,OptName:Longint;Var OptVal;optlen:longint) : Longint;`

Description: `GetSocketOptions` gets the connection options for socket `Sock`. The socket may be obtained from different levels, indicated by `Level`, which can be one of the following:

SOL_SOCKETFrom the socket itself.

XXXset `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

For more information on this call, refer to the unix manual page `getsockopt` (2) .

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULT`OptVal` points outside your address space.

See also: `GetSocketOptions` ([382](#))

Listen

Declaration: `Function Listen (Sock,MaxConnect:Longint) : Boolean;`

Description: `Listen` listens for up to `MaxConnect` connections from socket `Sock`. The socket `Sock` must be of type `SOCK_STREAM` or `Sock_SEQPACKET`. The function returns `True` if a connection was accepted, `False` if an error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EOPNOTSUPPThe socket type doesn't support the `Listen` operation.

See also: `Socket` ([385](#)), `Bind` ([378](#)), `Connect` ([379](#))

Recv

Declaration: `Function Recv (Sock:Longint;Var Addr;AddrLen,Flags:Longint) : Longint;`

Description: `Recv` reads at most `AddrLen` bytes from socket `Sock` into address `Addr`. The socket must be in a connected state. `Flags` can be one of the following:

- 1:** Process out-of band data.
- 4:** Bypass routing, use a direct interface.
- ??:** Wait for full request or report an error.

The function returns the number of bytes actually read from the socket, or -1 if a detectable error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

- SYS_EBADF**The socket descriptor is invalid.
- SYS_ENOTCONN**The socket isn't connected.
- SYS_ENOTSOCK**The descriptor is not a socket.
- SYS_EFAULT**The address is outside your address space.
- SYS EMSGSIZE**The message cannot be sent atomically.
- SYS_EWOULDBLOCK**The requested operation would block the process.
- SYS_ENOBUFS**The system doesn't have enough free buffers available.

See also: `Send` ([383](#))

Send

Declaration: `Function Send (Sock:Longint;Var Addr;AddrLen,Flags:Longint) : Longint;`

Description: `Send` sends `AddrLen` bytes starting from address `Addr` to socket `Sock`. `Sock` must be in a connected state. The function returns the number of bytes sent, or -1 if a detectable error occurred. `Flags` can be one of the following:

- 1:** Process out-of band data.
- 4:** Bypass routing, use a direct interface.

Errors: Errors are reported in `SocketError`, and include the following:

- SYS_EBADF**The socket descriptor is invalid.
- SYS_ENOTSOCK**The descriptor is not a socket.
- SYS_EFAULT**The address is outside your address space.
- SYS EMSGSIZE**The message cannot be sent atomically.
- SYS_EWOULDBLOCK**The requested operation would block the process.
- SYS_ENOBUFS**The system doesn't have enough free buffers available.

See also: `Recv` ([383](#)), `send` (2)

SetSocketOptions

Declaration: `Function SetSocketOptions (Sock,Level,OptName:Longint;Var OptVal;optlen:longint)
: Longint;`

Description: `SetSocketOptions` sets the connection options for socket `Sock`. The socket may be manipulated at different levels, indicated by `Level`, which can be one of the following:

SOL_SOCKETTo manipulate the socket itself.

XXXset `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

For more information on this call, refer to the unix manual page `setsockopt` (2) .

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULT`OptVal` points outside your address space.

See also: `GetSocketOptions` ([382](#))

Shutdown

Declaration: `Function Shutdown (Sock:Longint;How:Longint) : Longint;`

Description: `ShutDown` closes one end of a full duplex socket connection, described by `Sock`. `How` determines how the connection will be shut down, and can be one of the following:

0: Further receives are disallowed.

1: Further sends are disallowed.

2: Sending nor receiving are allowed.

On succes, the function returns 0, on error -1 is returned.

Errors: `SocketError` is used to report errors, and includes the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTCONNThe socket isn't connected.

SYS_ENOTSOCKThe descriptor is not a socket.

See also: `Socket` ([385](#)), `Connect` ([379](#))

Sock2File

Declaration: `Procedure Sock2File (Sock:Longint;Var SockIn,SockOut:File);`

Description: `Sock2File` transforms a socket `Sock` into 2 Pascal file descriptors of type `File`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: `Socket` ([385](#)), `Sock2Text` ([385](#))

Sock2Text

Declaration: `Procedure Sock2Text (Sock:Longint;Var SockIn,SockOut: Text);`

Description: `Sock2Text` transforms a socket `Sock` into 2 Pascal file descriptors of type `Text`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: `Socket` ([385](#)), `Sock2File` ([384](#))

Socket

Declaration: `Function Socket (Domain,SocketType,Protocol:Longint) : Longint;`

Description: `Socket` creates a new socket in domain `Domain`, from type `SocketType` using protocol `Protocol`. The `Domain`, `Socket` type and `Protocol` can be specified using predefined constants (see the section on constants for available constants) If succesfull, the function returns a socket descriptor, which can be passed to a subsequent `Bind` ([378](#)) call. If unsuccessful, the function returns -1.

Errors: Errors are returned in `SocketError`, and include the follwing:

SYS_EPROTONOSUPPORTThe protocol type or the specified protocol is not supported within this domain.

SYS_EMFILEThe per-process descriptor table is full.

SYS_ENFILEThe system file table is full.

SYS_EACCESSPermission to create a socket of the specified type and/or protocol is denied.

SYS_ENOBUFSInsufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

See also: `SocketPair` ([385](#)), `socket` (2)

for an example, see `Accept` ([376](#)).

SocketPair

Declaration: `Function SocketPair (Domain,SocketType,Protocol:Longint;var Pair:TSockArray) : Longint;`

Description: `SocketPair` creates 2 sockets in domain `Domain`, from type `SocketType` and using protocol `Protocol`. The pair is returned in `Pair`, and they are indistinguishable. The function returns -1 upon error and 0 upon success.

Errors: Errors are reported in `SocketError`, and are the same as in `Socket` ([385](#))

See also: `Str2UnixSockAddr` ([385](#))

Str2UnixSockAddr

Declaration: `Procedure Str2UnixSockAddr(const addr:string;var t:TUnixSockAddr;var len:longint)`

Description: `Str2UnixSockAddr` transforms a Unix socket address in a string to a `TUnixSockAddr` structure which can be passed to the `Bind` ([378](#)) call.

Errors: None.

See also: `Socket` ([385](#)), `Bind` ([378](#))

Chapter 21

The STRINGS unit.

This chapter describes the STRINGS unit for Free Pascal. This unit is system independent, and therefore works on all supported platforms.

Since the unit only provides some procedures and functions, there is only one section, which gives the declarations of these functions, together with an explanation.

21.1 Functions and procedures.

StrAlloc

Declaration: `Function StrAlloc (Len : Longint);PChar`

Description: StrAlloc reserves memory on the heap for a string with length Len, terminating #0 included, and returns a pointer to it.

Errors: If there is not enough memory, a run-time error occurs.

See also: StrNew ([393](#)), StrPCopy ([394](#)).

StrCat

Declaration: `Function StrCat (Dest,Source : PChar) : PChar;`

Description: Attaches Source to Dest and returns Dest.

Errors: No length checking is performed.

See also: Concat ()

```
Program Example11;  
  
Uses strings;  
  
{ Program to demonstrate the StrCat function. }  
  
Const P1 : PChar = 'This is a PChar String.';  
  
Var P2 : PChar;  
  
begin
```

```
P2:= StrAlloc (StrLen(P1)*2+1);
StrMove (P2,P1,StrLen(P1)+1); { P2=P1 }
StrCat (P2,P1);                { Append P2 once more }
Writeln ('P2 : ',P2);
end.
```

StrComp

Declaration: Function StrComp (S1,S2 : PChar) : Longint;

Description: Compares the null-terminated strings S1 and S2. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: StrLComp ([390](#)), StrIComp ([389](#)), StrLComp ([392](#))

For an example, see StrLComp ([390](#)).

StrCopy

Declaration: Function StrCopy (Dest,Source : PChar) : PChar;

Description: Copy the null terminated string in Source to Dest, and returns a pointer to Dest. Dest needs enough room to contain Source, i.e. StrLen(Source)+1 bytes.

Errors: No length checking is performed.

See also: StrPCopy ([394](#)), StrLCopy ([391](#)), StrECopy ([388](#))

```
Program Example4;

Uses strings;

{ Program to demonstrate the StrCopy function. }

Const P : PChar = 'This is a PCHAR string.';

var PP : PChar;

begin
  PP:= StrAlloc (StrLen(P)+1);
  StrCopy (PP,P);
  If StrComp (PP,P)<>0 then
    Writeln ('Oh-oh problems...')
  else
    Writeln ('All is well : PP=',PP);
end.
```

StrDispose

Declaration: `Procedure StrDispose (P : PChar);`

Description: Removes the string in P from the heap and releases the memory.

Errors: None.

See also: `Dispose ()`, `StrNew` ([393](#))

```
Program Example17;

Uses strings;

{ Program to demonstrate the StrDispose function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  Writeln ('Before StrNew : Memory available : ', MemAvail);
  P2:=StrNew (P1);
  Writeln ('After StrNew : Memory available : ', MemAvail);
  Writeln ('P2 : ', P2);
  StrDispose(P2);
  Writeln ('After StrDispose : Memory available : ', MemAvail);
end.
```

StrECopy

Declaration: `Function StrECopy (Dest,Source : PChar) : PChar;`

Description: Copies the Null-terminated string in Source to Dest, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: `StrLCopy` ([391](#)), `StrCopy` ([387](#))

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin
  PP:=StrAlloc (StrLen(P)+1);
  If Longint(StrECopy(PP,P))–Longint(PP)<>StrLen(P) then
    Writeln('Something is wrong here !')
  else
    Writeln ('PP= ',PP);
end.
```

StrEnd

Declaration: `Function StrEnd (P : PChar) : PChar;`

Description: Returns a pointer to the end of P. (i.e. to the terminating null-character.

Errors: None.

See also: [StrLen \(391\)](#)

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';

begin
  If Longint(StrEnd(P)) - Longint(P) <> StrLen(P) then
    WriteLn('Something is wrong here !')
  else
    WriteLn('All is well..');
end.
```

StrIComp

Declaration: `Function StrIComp (S1,S2 : PChar) : Longint;`

Description: Compares the null-terminated strings S1 and S2, ignoring case. The result is

- A negative Longint when S1 < S2.
- 0 when S1 = S2.
- A positive Longint when S1 > S2.

Errors: None.

See also: [StrLComp \(390\)](#), [StrComp \(387\)](#), [StrLIComp \(392\)](#)

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PChar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp(P1,P2) <> 0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(L);
  WriteLn (L, ' characters are the same.');
```

StrLCat

Declaration: Function StrLCat (Dest,Source : PChar; MaxLen : Longint) : PChar;

Description: Adds MaxLen characters from Source to Dest, and adds a terminating null-character. Returns Dest.

Errors: None.

See also: StrCat ([386](#))

```
Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin
  P2:= StrAlloc (StrLen(P1)*2+1);
  P2^:=#0; { Zero length }
  StrCat (P2,P1);
  StrLCat (P2,P1,5);
  Writeln ('P2 = ',P2);
end.
```

StrLComp

Declaration: Function StrLComp (S1,S2 : PChar; L : Longint) : Longint;

Description: Compares maximum L characters of the null-terminated strings S1 and S2. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: StrComp ([387](#)), StrlComp ([389](#)), StrLIComp ([392](#))

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PChar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
```



```
L:=1;  
While StrLComp(P1,P2,L)=0 do inc (L);  
dec(L);  
Writeln (L,' characters are the same.');
```

```
end.
```

StrLCopy

Declaration: Function StrLCopy (Dest,Source : PChar; MaxLen : Longint) : PChar;

Description: Copies MaxLen characters from Source to Dest, and makes Dest a null terminated string.

Errors: No length checking is performed.

See also: StrCopy ([387](#)), StrECopy ([388](#))

```
Program Example5;  
  
Uses strings;  
  
{ Program to demonstrate the StrLCopy function. }  
  
Const P : PChar = '123456789ABCDEF';  
  
var PP : PChar;  
  
begin  
  PP:= StrAlloc(11);  
  Writeln ('First 10 characters of P : ',StrLCopy (PP,P,10));  
end.
```

StrLen

Declaration: Function StrLen (p : PChar) : Longint;

Description: Returns the length of the null-terminated string P.

Errors: None.

See also: Length ()

```
Program Example1;  
  
Uses strings;  
  
{ Program to demonstrate the StrLen function. }  
  
Const P : PChar = 'This is a constant pchar string';  
  
begin  
  Writeln ('P          : ',p);  
  Writeln ('length(P) : ',StrLen(P));  
end.
```

StrLComp

Declaration: `Function StrLComp (S1,S2 : PChar; L : Longint) : Longint;`

Description: Compares maximum L characters of the null-terminated strings S1 and S2, ignoring case. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: [StrLComp \(390\)](#), [StrComp \(387\)](#), [StrComp \(389\)](#)

For an example, see [StrComp \(389\)](#)

StrLower

Declaration: `Function StrLower (P : PChar) : PChar;`

Description: Converts P to an all-lowercase string. Returns P.

Errors: None.

See also: [Upcase \(\)](#) , [StrUpper \(395\)](#)

Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

Const

P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';
P2 : PChar = 'this is a lowercase string';

begin

WriteIn ('Uppercase : ',**StrUpper**(P2));

StrLower (P1);

WriteIn ('Lowercase : ',P1);

end.

StrMove

Declaration: `Function StrMove (Dest,Source : PChar; MaxLen : Longint) : PChar;`

Description: Copies MaxLen characters from Source to Dest. No terminating null-character is copied. Returns Dest.

Errors: None.

See also: [StrLCopy \(391\)](#), [StrCopy \(387\)](#)

```
Program Example10;  
  
Uses strings;  
  
{ Program to demonstrate the StrMove function. }  
  
Const P1 : PCHAR = 'This is a pchar string.';  
  
Var P2 : Pchar;  
  
begin  
  P2:=StrAlloc(StrLen(P1)+1);  
  StrMove (P2,P1,StrLen(P1)+1); { P2:=P1 }  
  Writeln ('P2 = ',P2);  
end.
```

StrNew

Declaration: `Function StrNew (P : PChar) : PChar;`

Description: Copies P to the Heap, and returns a pointer to the copy.

Errors: Returns Nil if no memory was available for the copy.

See also: `New()`, `StrCopy` ([387](#)), `StrDispose` ([388](#))

```
Program Example16;  
  
Uses strings;  
  
{ Program to demonstrate the StrNew function. }  
  
Const P1 : PChar = 'This is a PChar string';  
  
var P2 : PChar;  
  
begin  
  P2:=StrNew (P1);  
  If P1=P2 then  
    writeln ('This can''t be happening...')  
  else  
    writeln ('P2 : ',P2);  
end.
```

StrPas

Declaration: `Function StrPas (P : PChar) : String;`

Description: Converts a null terminated string in P to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

See also: `StrPCopy` ([394](#))

```
Program Example3;  
  
Uses strings;  
  
{ Program to demonstrate the StrPas function. }  
  
Const P : PChar = 'This is a PCHAR string';  
  
var S : string;  
  
begin  
  S:=StrPas (P);  
  Writeln ('S : ',S);  
end.
```

StrPCopy

Declaration: `Function StrPCopy (Dest : PChar; Const Source : String) : PChar;`

Description: Converts the Pascal string in Source to a Null-terminated string, and copies it to Dest. Dest needs enough room to contain the string Source, i.e. `Length(Source)+1` bytes.

Errors: No length checking is performed.

See also: [StrPas \(393\)](#)

```
Program Example2;  
  
Uses strings;  
  
{ Program to demonstrate the StrPCopy function. }  
  
Const S = 'This is a normal string.';  
  
Var P : Pchar;  
  
begin  
  p:=StrAlloc (length(S)+1);  
  if StrPCopy (P,S)<>P then  
    Writeln ('This is impossible !!')  
  else  
    writeln (P);  
end.
```

StrPos

Declaration: `Function StrPos (S1,S2 : PChar) : PChar;`

Description: Returns a pointer to the first occurrence of S2 in S1. If S2 does not occur in S1, returns Nil.

Errors: None.

See also: [Pos \(\)](#) , [StrScan \(395\)](#), [StrRScan \(395\)](#)

```
Program Example15;
```

Uses strings;

{ Program to demonstrate the StrPos function. }

Const P : PChar = 'This is a PChar string.';
 S : PChar = 'is';

begin

Writeln ('Position of ''is'' in P : ', longint(**StrPos**(P,S)) – Longint(P));
end.

StrRScan

Declaration: Function StrRScan (P : PChar; C : Char) : PChar;

Description: Returns a pointer to the last occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

Errors: None.

See also: Pos (), StrScan ([395](#)), StrPos ([394](#))

For an example, see StrScan ([395](#)).

StrScan

Declaration: Function StrScan (P : PChar; C : Char) : PChar;

Description: Returns a pointer to the first occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

Errors: None.

See also: Pos (), StrRScan ([395](#)), StrPos ([394](#))

Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

Const P : PChar = 'This is a PCHAR string.';
 S : Char = 's' ;

begin

Writeln ('P, starting from first ''s'' : ', **StrScan**(P,s));
 Writeln ('P, starting from last ''s'' : ', **StrRScan**(P,s));
end.

StrUpper

Declaration: Function StrUpper (P : PChar) : PChar;

Description: Converts P to an all-uppercase string. Returns P.

Errors: None.

See also: `Uppcase ()` , `StrLower` ([392](#))

For an example, see `StrLower` ([392](#))

Chapter 22

The SYSUTILS unit.

This chapter describes the `sysutils` unit. The `sysutils` unit was largely written by Gertjan Schouten, and completed by Michael Van Canneyt. It aims to be compatible to the Delphi `sysutils` unit, but in contrast with the latter, it is designed to work on multiple platforms. It is implemented on all supported platforms.

This chapter starts out with a definition of all types and constants that are defined, followed by an overview of functions grouped by functionality, and lastly the complete explanation of each function.

22.1 Constants and types

The following general-purpose types are defined:

```
tfilename = string;

tsyscharset = set of char;
tintegerset = set of 0..sizeof(integer)*8-1;

longrec = packed record
    lo,hi : word;
end;

wordrec = packed record
    lo,hi : byte;
end;

TMethod = packed record
    Code, Data: Pointer;
end;
```

The use and meaning of these types should be clear, so no extra information will be provided here.

The following general-purpose constants are defined:

```
const
    SecsPerDay = 24 * 60 * 60; // Seconds and milliseconds per day
    MSecsPerDay = SecsPerDay * 1000;
    DateDelta = 693594;        // Days between 1/1/0001 and 12/31/1899
    Eoln = #10;
```

The following types are used frequently in date and time functions. They are the same on all platforms.

```
type
  TSystemTime = record
    Year, Month, Day: word;
    Hour, Minute, Second, MilliSecond: word;
  end ;

  TDateTime = double;

  TTimeStamp = record
    Time: integer;    { Number of milliseconds since midnight }
    Date: integer;    { One plus number of days since 1/1/0001 }
  end ;
```

The following type is used in the [FindFirst \(434\)](#), [FindNext \(434\)](#) and [FindClose \(433\)](#) functions. The win32 version differs from the other versions. If code is to be portable, that part shouldn't be used.

```
Type
  THandle = Longint;
  TSearchRec = Record
    Time, Size, Attr : Longint;
    Name : TFileName;
    ExcludeAttr : Longint;
    FindHandle : THandle;
    {$ifdef Win32}
    FindData : TWin32FindData;
    {$endif}
  end;
```

The following constants are file-attributes that need to be matched in the findfirst call.

```
Const
  faReadOnly    = $00000001;
  faHidden      = $00000002;
  faSysFile     = $00000004;
  faVolumeId    = $00000008;
  faDirectory   = $00000010;
  faArchive     = $00000020;
  faAnyFile     = $0000003f;
```

The following constants can be used in the [FileOpen \(430\)](#) call.

```
Const
  fmOpenRead      = $0000;
  fmOpenWrite     = $0001;
  fmOpenReadWrite = $0002;
```

The following constants can be used in the [FileSeek \(432\)](#) call.

```
Const
  fsFromBeginning = 0;
```



```
fsFromCurrent    = 1;
fsFromEnd        = 2;
```

The following variables are used in the case translation routines.

```
type
  TCaseTranslationTable = array[0..255] of char;
var
  UpperCaseTable: TCaseTranslationTable;
  LowerCaseTable: TCaseTranslationTable;
```

The initialization code of the `sysutils` unit fills these tables with the appropriate values. For the win32 and go32v2 versions, this information is obtained from the operating system.

The following constants control the formatting of dates. For the Win32 version of the `sysutils` unit, these constants are set according to the internationalization settings of Windows by the initialization code of the unit.

```
Const
  DateSeparator: char = '-';
  ShortDateFormat: string = 'd/m/y';
  LongDateFormat: string = 'dd" "mmmm" "yyyy';
  ShortMonthNames: array[1..12] of string[128] =
    ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
     'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec');
  LongMonthNames: array[1..12] of string[128] =
    ('January', 'February', 'March', 'April',
     'May', 'June', 'July', 'August',
     'September', 'October', 'November', 'December');
  ShortDayNames: array[1..7] of string[128] =
    ('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat');
  LongDayNames: array[1..7] of string[128] =
    ('Sunday', 'Monday', 'Tuesday', 'Wednesday',
     'Thursday', 'Friday', 'Saturday');
```

The following constants control the formatting of times. For the Win32 version of the `sysutils` unit, these constants are set according to the internationalization settings of Windows by the initialization code of the unit.

```
Const
  ShortTimeFormat: string = 'hh:nn';
  LongTimeFormat: string = 'hh:nn:ss';
  TimeSeparator: char = ':';
  TimeAMString: string[7] = 'AM';
  TimePMString: string[7] = 'PM';
```

The following constants control the formatting of currencies and numbers. For the Win32 version of the `sysutils` unit, these constants are set according to the internationalization settings of Windows by the initialization code of the unit.

```
Const
  DecimalSeparator : Char = '.';
  ThousandSeparator : Char = ',';
  CurrencyDecimals : Byte = 2;
```

```
CurrencyString : String[7] = '$';
{ Format to use when formatting currency :
  0 = $1          1 = 1$          2 = $ 1          3 = 1 $
  4 = Currency string replaces decimal indicator.
      e.g. 1$50
}
CurrencyFormat : Byte = 1;
{ Same as above, only for negative currencies:
  0 = ($1)
  1 = -$1
  2 = $-1
  3 = $1-
  4 = (1$)
  5 = -1$
  6 = 1-$
  7 = 1$-
  8 = -1 $
  9 = -$ 1
  10 = $ 1-
}
NegCurrFormat : Byte = 5;
```

The following types are used in various string functions.

```
type
  PString = ^String;
  TFloatFormat = (ffGeneral, ffExponent, ffFixed, ffNumber, ffCurrency);
```

The following constants are used in the file name handling routines. Do not use a slash or backslash character directly as a path separator; instead use the `OsDirSeparator` character.

```
Const
  DirSeparators : set of char = ['/', '\'];
{$ifdef unix}
  OsDirSeparator = '/';
{$else}
  OsDirSeparator = '\';
{$endif}
```

22.2 Function list by category

What follows is a listing of the available functions, grouped by category. For each function there is a reference to the page where you can find the function.

String functions

Functions for handling strings.

Name	Description	Page
<code>AnsiCompareStr</code>	Compare two strings	439
<code>AnsiCompareText</code>	Compare two strings, case insensitive	440

AnsiExtractQuotedStr	Removes quotes from string	441
AnsiLastChar	Get last character of string	442
AnsiLowerCase	Convert string to all-lowercase	442
AnsiQuotedStr	Quotes a string	443
AnsiStrComp	Compare strings case-sensitive	443
AnsiStrlComp	Compare strings case-insensitive	444
AnsiStrLComp	Compare L characters of strings case sensitive	445
AnsiStrLIComp	Compare L characters of strings case insensitive	446
AnsiStrLastChar	Get last character of string	445
AnsiStrLower	Convert string to all-lowercase	447
AnsiStrUpper	Convert string to all-uppercase	448
AnsiUpperCase	Convert string to all-uppercase	448
AppendStr	Append 2 strings	449
AssignStr	Assign value of strings on heap	449
CompareStr	Compare two strings case sensitive	451
CompareText	Compare two strings case insensitive	451
DisposeStr	Remove string from heap	452
IsValidIdent	Is string a valid pascal identifier	463
LastDelimiter	Last occurrence of character in a string	464
LeftStr	Get first N characters of a string	464
LoadStr	Load string from resources	464
LowerCase	Convert string to all-lowercase	465
NewStr	Allocate new string on heap	465
RightStr	Get last N characters of a string	466
StrAlloc	Allocate memory for string	437
StrBufSize	Reserve memory for a string	437
StrDispose	Remove string from heap	438
StrPas	Convert PChar to pascal string	439
StrPCopy	Copy pascal string	438
StrPLCopy	Copy N bytes of pascal string	438
UpperCase	Convert string to all-uppercase	470

Formatting strings

Functions for formatting strings.

Name	Description	Page
AdjustLineBreaks	Convert line breaks to line breaks for system	439
FormatBuf	Format a buffer	461
Format	Format arguments in string	456
FmtStr	Format buffer	456

QuotedStr	Quote a string	465
StrFmt	Format arguments in a string	466
StrLFmt	Format maximum L characters in a string	467
TrimLeft	Remove whitespace at the left of a string	469
TrimRight	Remove whitespace at the right of a string	470
Trim	Remove whitespace at both ends of a string	468

File input/output routines

Functions for reading/writing to file.

Name	Description	Page
FileCreate	Create a file and return handle	427
FileOpen	Open file and return handle	430
FileRead	Read from file	431
FileSeek	Set file position	432
FileTruncate	Truncate file length	433
FileWrite	Write to file	433
FileClose	Close file handle	427

File handling routines

Functions for file manipulation.

Name	Description	Page
AddDisk	Add disk to list of disk drives	419
ChangeFileExt	Change extension of file name	422
CreateDir	Create a directory	419
DeleteFile	Delete a file	422
DiskFree	Free space on disk	420
DiskSize	Total size of disk	420
ExpandFileName	Create full file name	423
ExpandUNCFileName	Create full UNC file name	424
ExtractFileDir	Extract directory part of filename	424
ExtractFileDrive	Extract drive part of filename	425
ExtractFileExt	Extract extension part of filename	425
ExtractFileName	Extract name part of filename	425
ExtractFilePath	Extract path part of filename	426
ExtractRelativePath	Construct relative path between two files	426
FileAge	Return file age	427
FileDateToDateTime	Convert file date to system date	411
FileExists	Determine whether a file exists on disk	428

FileGetAttr	Get attributes of file	429
FileGetDate	Get date of last file modification	430
FileSearch	Search for file in path	431
FileSetAttr	Get file attributes	432
FileSetDate	Get file dates	433
FindFirst	Start finding a file	434
FindNext	Find next file	434
GetCurrentDir	Return current working directory	421
RemoveDir	Remove a directory from disk	421
RenameFile	Rename a file on disk	435
SetCurrentDir	Set current working directory	422
SetDirSeparators	Set directory separator characters	436
FindClose	Stop searching a file	433
DoDirSeparators	Replace directory separator characters	423

Date/time routines

Functions for date and time handling.

Name	Description	Page
DateTimeToFileDate	Convert DateTime type to file date	406
DateTimeToStr	Construct string representation of DateTime	406
DateTimeToString	Construct string representation of DateTime	407
DateTimeToSystemTime	Convert DateTime to system time	408
DateTimeToTimeStamp	Convert DateTime to timestamp	408
DateToStr	Construct string representation of date	409
Date	Get current date	405
DayOfWeek	Get day of week	409
DecodeDate	Decode DateTime to year month and day	409
DecodeTime	Decode DateTime to hours, minutes and seconds	410
EncodeDate	Encode year, day and month to DateTime	410
EncodeTime	Encode hours, minutes and seconds to DateTime	411
FormatDateTime	Return string representation of DateTime	412
IncMonth	Add 1 to month	412
IsLeapYear	Determine if year is leap year	413
MSecsToTimeStamp	Convert nr of milliseconds to timestamp	414
Now	Get current date and time	414
StrToDateTime	Convert string to DateTime	415
StrToDate	Convert string to date	415
StrToTime	Convert string to time	416
SystemTimeToDateTime	Convert system time to datetime	417

TimeStampToDateTime	Convert time stamp to DateTime	418
TimeStampToMSecs	Convert Timestamp to number of milliseconds	418
TimeToStr	return string representation of Time	418
Time	Get current tyme	417

22.3 Miscellaneous conversion routines

Functions for various conversions.

Name	Description	Page
BCDToInt	Convert BCD number to integer	450
CompareMem	Compare two memory regions	450
FloatToStrF	Convert float to formatted string	453
FloatToStr	Convert float to string	453
FloatToText	Convert float to string	455
GetDirs	Split string in list of directories	435
IntToHex	return hexadecimal representation of integer	462
IntToStr	return decumal representation of integer	462
StrToIntDef	Convert string to integer with default value	468
StrToInt	Convert string to integer	467

22.4 Date and time functions

Date and time formatting characters

Various date and time formatting routines accept a format string. to format the date and or time. The following characters can be used to control the date and time formatting:

c : shortdateformat + ' ' + shorttimeformat

d : day of month

dd : day of month (leading zero)

ddd : day of week (abbreviation)

dddd : day of week (full)

dddddd : shortdateformat

ddddddd : longdateformat

m : month

mm : month (leading zero)

mmm : month (abbreviation)

mmmm : month (full)

y : year (four digits)
yy : year (two digits)
yyyy : year (with century)
h : hour
hh : hour (leading zero)
n : minute
nn : minute (leading zero)
s : second
ss : second (leading zero)
t : shorttimeformat
tt : longtimeformat
am/pm : use 12 hour clock and display am and pm accordingly
a/p : use 12 hour clock and display a and p accordingly
/ : insert date separator
: : insert time separator
"xx" : literal text
'xx' : literal text

TDateTime

Declaration: `TDateTime = Double;`

Description: Many functions return or require a `TDateTime` type, which contains a date and time in encoded form. The date and time are converted to a double as follows:

- The date part is stored in the integer part of the double as the number of days passed since January 1, 1900.
- The time part is stored in the fractional part of the double, as the number of milliseconds passed since midnight (00:00), divided by the total number of milliseconds in a day.

Date

Declaration: `Function Date: TDateTime;`

Description: `Date` returns the current date in `TDateTime` format. For more information about the `TDateTime` type, see `TDateTime` (405).

Errors: None.

See also: `Time` (417), `Now` (414), `TDateTime` (405).

Listing: `sysutex/ex1.pp`

```
Program Example1;

{ This program demonstrates the Date function }

uses sysutils;

Var YY,MM,DD : Word;

Begin
  WriteLn ( 'Date : ', Date );
  DecodeDate ( Date,YY,MM,DD);
  WriteLn ( format ( 'Date is (DD/MM/YY): %d/%d/%d ',[dd,mm,yy]));
End.
```

DateTimeToFileDate

Declaration: `Function DateTimeToFileDate(DateTime : TDateTime) : Longint;`

Description: `DateTimeToFileDate` function converts a date/time indication in `TDateTime` format to a filedate function, such as returned for instance by the `FileAge` ([427](#)) function.

Errors: None.

See also: `Time` ([417](#)), `Date` ([405](#)), `FileDateToDateTime` ([411](#)), `DateTimeToSystemTime` ([408](#)), `DateTimeToTimeStamp` ([408](#))

Listing: `sysutex/ex2.pp`

```
Program Example2;

{ This program demonstrates the DateTimeToFileDate function }

Uses sysutils;

Begin
  WriteLn ( 'FileTime of now would be: ', DateTimeToFileDate (Now));
End.
```

DateTimeToStr

Declaration: `Function DateTimeToStr(DateTime: TDateTime): string;`

Description: `DateTimeToStr` returns a string representation of `DateTime` using the formatting specified in `ShortDateTimeFormat`. It corresponds to a call to `FormatDateTime('c',DateTime)` (see section [22.4](#), page [404](#)).

Errors: None.

See also: `FormatDateTime` ([412](#)), `TDateTime` ([405](#)).

Listing: `sysutex/ex3.pp`

```
Program Example3;

{ This program demonstrates the DateTimeToStr function }
```


Uses sysutils;

Begin

Writeln ('Today is : ', **DateTimeToStr**(**Now**));

Writeln ('Today is : ', **FormatDateTime**('c',**Now**));

End.

DateTimeToString

Declaration: `Procedure DateTimeToString(var Result: string; const FormatStr: string; const DateTime: TDateTime);`

Description: `DateTimeToString` returns in `Result` a string representation of `DateTime` using the formatting specified in `FormatStr`.

for a list of characters that can be used in the `FormatStr` formatting string, see section 22.4, page 404.

Errors: In case a wrong formatting character is found, an `EConvertError` is raised.

See also: `FormatDateTime` (412), section 22.4, page 404.

Listing: sysutex/ex4.pp

Program Example4;

{ This program demonstrates the DateTimeToString function }

Uses sysutils;

Procedure today (Fmt : **string**);

Var S : **AnsiString**;

begin

DateTimeToString (S,Fmt,**Date**);

Writeln (S);

end;

Procedure Now (Fmt : **string**);

Var S : **AnsiString**;

begin

DateTimeToString (S,Fmt,**Time**);

Writeln (S);

end;

Begin

 Today ('"Today is "dddd dd mmmm y');

 Today ('"Today is "d mmm yy');

 Today ('"Today is "d/mmm/yy');

Now (' ' 'The time is ' 'am/pmh:n:s');

Now (' ' 'The time is ' 'hh:nn:ssam/pm');

Now (' ' 'The time is ' 'tt');

End.

DateTimeToSystemTime

Declaration: Procedure DateTimeToSystemTime(DateTime: TDateTime; var SystemTime: TSystemTime);

Description: DateTimeToSystemTime converts a date/time pair in DateTime, with TDateTime format to a system time SystemTime.

Errors: None.

See also: DateTimeToFileDate ([406](#)), SystemTimeToDateTime ([417](#)), DateTimeToTimeStamp ([408](#))

Listing: sysutex/ex5.pp

Program Example5;

{ This program demonstrates the DateTimeToSystemTime function }

Uses sysutils;

Var ST : TSystemTime;

Begin

DateTimeToSystemTime(**Now**, ST);

With St **do**

begin

WriteLn ('Today is ', year, '/', month, '/', Day);

WriteLn ('The time is ', Hour, ':', minute, ':', Second, '.', MilliSecond);

end;

End.

DateTimeToTimeStamp

Declaration: Function DateTimeToTimeStamp(DateTime: TDateTime): TTimeStamp;

Description: DateTimeToSystemTime converts a date/time pair in DateTime, with TDateTime format to a TTimeStamp format.

Errors: None.

See also: DateTimeToFileDate ([406](#)), SystemTimeToDateTime ([417](#)), DateTimeToSystemTime ([408](#))

Listing: sysutex/ex6.pp

Program Example6;

{ This program demonstrates the DateTimeToTimeStamp function }

Uses sysutils;

Var TS : TTimeStamp;

Begin

TS:=DateTimeToTimeStamp (**Now**);

With TS **do**

begin

WriteLn ('Now is ', **time**, ' millisecond past midnight');

WriteLn ('Today is ', **Date**, ' days past 1/1/0001');

end;

End.

DateToStr

Declaration: `Function DateToStr(Date: TDateTime): string;`

Description: `DateToStr` converts `Date` to a string representation. It uses `ShortDateFormat` as its formatting string. It is hence completely equivalent to a `FormatDateTime('dddd', Date)`.

Errors: None.

See also: `TimeToStr` ([418](#)), `DateTimeToStr` ([406](#)), `FormatDateTime` ([412](#)), `StrToDate` ([415](#))

Listing: `sysutex/ex7.pp`

Program `Example7;`

{ This program demonstrates the DateToStr function }

Uses `sysutils;`

Begin

`WriteLn (Format ('Today is: %s', [DateToStr(Date)]));`

End.

DayOfWeek

Declaration: `Function DayOfWeek(DateTime: TDateTime): integer;`

Description: `DayOfWeek` returns the day of the week from `DateTime`. Sunday is counted as day 1, Saturday is counted as day 7. The result of `DayOfWeek` can serve as an index to the `LongDayNames` constant array, to retrieve the name of the day.

Errors: None.

See also: `Date` ([405](#)), `DateToStr` ([409](#))

Listing: `sysutex/ex8.pp`

Program `Example8;`

{ This program demonstrates the DayOfWeek function }

Uses `sysutils;`

Begin

`WriteLn ('Today''s day is ', LongDayNames[DayOfWeek(Date)]);`

End.

DecodeDate

Declaration: `Procedure DecodeDate(Date: TDateTime; var Year, Month, Day: word);`

Description: `DecodeDate` decodes the Year, Month and Day stored in `Date`, and returns them in the Year, Month and Day variables.

Errors: None.

See also: `EncodeDate` ([410](#)), `DecodeTime` ([410](#)).

Listing: sysutex/ex9.pp

Program Example9;

{ This program demonstrates the DecodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin

DecodeDate(**Date**,YY,MM,DD);

WriteLn (**Format** ('Today is %d/%d/%d' ,[dd,mm,yy]));

End.

DecodeTime

Declaration: Procedure DecodeTime(Time: TDateTime; var Hour, Minute, Second, MilliSecond: word);

Description: DecodeDate decodes the hours, minutes, second and milliseconds stored in Time, and returns them in the Hour, Minute and Second and MilliSecond variables.

Errors: None.

See also: EncodeTime ([411](#)), DecodeDate ([409](#)).

Listing: sysutex/ex10.pp

Program Example10;

{ This program demonstrates the DecodeTime function }

Uses sysutils;

Var HH,MM,SS,MS: Word;

Begin

DecodeTime(**Time**,HH,MM,SS,MS);

WriteLn (**format**('The time is %d:%d:%d.%d' ,[hh,mm,ss,ms]));

End.

EncodeDate

Declaration: Function EncodeDate(Year, Month, Day :word): TDateTime;

Description: EncodeDate encodes the Year, Month and Day variables to a date in TDateTime format. It does the opposite of the DecodeDate ([409](#)) procedure.

The parameters must lie withing valid ranges (boundaries included):

Year must be between 1 and 9999.

Month must be within the range 1-12.

Days must be between 1 and 31.

Errors: In case one of the parameters is out of it's valid range, 0 is returned.

See also: [EncodeTime \(411\)](#), [DecodeDate \(409\)](#).

Listing: sysutex/ex11.pp

Program Example11;

{ This program demonstrates the EncodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin

DecodeDate (**Date**,YY,MM,DD);

WriteLn ('Today is : ',**FormatDateTime** ('dd mmm yyyy ',**EnCodeDate**(YY,Mm,Dd)));

End.

EncodeTime

Declaration: Function EncodeTime(Hour, Minute, Second, MilliSecond:word): TDateTime;

Description: EncodeTime encodes the Hour, Minute, Second, MilliSecond variables to a TDateTime format result. It does the opposite of the [DecodeTime \(410\)](#) procedure.

The parameters must have a valid range (boundaries included):

Hour must be between 0 and 23.

Minute,second must both be between 0 and 59.

Millisecond must be between 0 and 999.

Errors: In case one of the parameters is outside of it's valid range, 0 is returned.

See also: [EncodeDate \(410\)](#), [DecodeTime \(410\)](#).

Listing: sysutex/ex12.pp

Program Example12;

{ This program demonstrates the EncodeTime function }

Uses sysutils;

Var Hh,MM,SS,MS : Word;

Begin

DeCodeTime (**Time**,Hh,MM,SS,MS);

WriteLn ('Present Time is : ',**FormatDateTime** ('hh:mm:ss ',**EnCodeTime** (Hh,MM,SS,MS)));

End.

FileDateToDateTime

Declaration: Function FileDateToDateTime(Filedate : Longint) : TDateTime;

Description: FileDateToDateTime converts the date/time encoded in filedate to a TDateTime encoded form. It can be used to convert date/time values returned by the [FileAge \(427\)](#) or [FindFirst \(434\)](#)/[FindNext \(434\)](#) functions to TDateTime form.

Errors: None.

See also: [DateTimeToFileDate \(406\)](#)

Listing: sysutex/ex13.pp

Program Example13;

{ This program demonstrates the FileDateToDateTime function }

Uses sysutils;

Var

 ThisAge : Longint;

Begin

Write ('ex13.pp created on :');

 ThisAge := **FileAge** ('ex13.pp');

Writeln (**DateTimeToStr** (**FileDateToDateTime** (ThisAge)));

End.

FormatDateTime

Declaration: `Function FormatDateTime(FormatStr: string; DateTime: TDateTime):string;`

Description: `FormatDateTime` formats the date and time encoded in `DateTime` according to the formatting given in `FormatStr`. The complete list of formatting characters can be found in section [22.4](#), page [404](#).

Errors: On error (such as an invalid character in the formatting string), and `EConvertError` exception is raised.

See also: [DateTimeToStr \(406\)](#), [DateToStr \(409\)](#), [TimeToStr \(418\)](#), [StrToDateTime \(415\)](#)

Listing: sysutex/ex14.pp

Program Example14;

{ This program demonstrates the FormatDateTime function }

Uses sysutils;

Var ThisMoment : TDateTime;

Begin

 ThisMoment := **Now**;

Writeln ('Now : ', **FormatDateTime** ('hh:mm', ThisMoment));

Writeln ('Now : ', **FormatDateTime** ('DD MM YYYY', ThisMoment));

Writeln ('Now : ', **FormatDateTime** ('c', ThisMoment));

End.

IncMonth

Declaration: `Function IncMonth(const DateTime: TDateTime; NumberOfMonths: integer): TDateTime;`

Description: `IncMonth` increases the month number in `DateTime` with `NumberOfMonths`. It wraps the result as to get a month between 1 and 12, and updates the year accordingly. `NumberOfMonths` can be negative, and can be larger than 12 (in absolute value).

Errors: None.

See also: `Date` ([405](#)), `Time` ([417](#)), `Now` ([414](#))

Listing: `sysutex/ex15.pp`

Program `Example15`;

{ This program demonstrates the IncMonth function }

Uses `sysutils`;

Var `ThisDay` : `TDateTime`;

Begin

`ThisDay := Date`;

`WriteLn ('ThisDay : ', DateToStr(ThisDay));`

`WriteLn ('6 months ago : ', DateToStr(IncMonth(ThisDay, -6)));`

`WriteLn ('6 months from now : ', DateToStr(IncMonth(ThisDay, 6)));`

`WriteLn ('12 months ago : ', DateToStr(IncMonth(ThisDay, -12)));`

`WriteLn ('12 months from now : ', DateToStr(IncMonth(ThisDay, 12)));`

`WriteLn ('18 months ago : ', DateToStr(IncMonth(ThisDay, -18)));`

`WriteLn ('18 months from now : ', DateToStr(IncMonth(ThisDay, 18)));`

End.

IsLeapYear

Declaration: `Function IsLeapYear(Year: Word): boolean`;

Description: `IsLeapYear` returns `True` if `Year` is a leap year, `False` otherwise.

Errors: None.

See also: `IncMonth` ([412](#)), `Date` ([405](#))

Listing: `sysutex/ex16.pp`

Program `Example16`;

{ This program demonstrates the IsLeapYear function }

Uses `sysutils`;

Var `YY, MM, dd` : `Word`;

Procedure `TestYear (Y : Word)`;

begin

`WriteLn (Y, ' is leap year : ', IsLeapYear(Y));`

end;

Begin

`DeCodeDate (Date, YY, mm, dd);`

`TestYear (yy);`

```
TestYear(2000);
TestYear(1900);
TestYear(1600);
TestYear(1992);
TestYear(1995);
End.
```

MSecsToTimeStamp

Declaration: `Function MSecsToTimeStamp(MSecs: Comp): TTimeStamp;`

Description: `MSecsToTimeStamp` converts the given number of milliseconds to a `TTimeStamp` date/time notation.

Use `TTimeStamp` variables if you need to keep very precise track of time.

Errors: None.

See also: `TimeStampToMSecs` ([418](#)), `DateTimeToTimeStamp` ([408](#)),

Listing: `sysutex/ex17.pp`

Program `Example17;`

{ This program demonstrates the MSecsToTimeStamp function }

Uses `sysutils;`

Var `MS : Comp;`
 `TS : TTimeStamp;`
 `DT : TDateTime;`

Begin

```
TS:=DateTimeToTimeStamp(Now);
WriteLn ('Now in days since 1/1/0001      : ',TS.Date);
WriteLn ('Now in millisecs since midnight : ',TS.Time);
MS:=TimeStampToMSecs(TS);
WriteLn ('Now in millisecs since 1/1/0001 : ',MS);
MS:=MS-1000*3600*2;
TS:=MSecsToTimeStamp(MS);
DT:=TimeStampToDateTime(TS);
WriteLn ('Now minus 1 day : ',DateTimeToStr(DT));
```

End.

Now

Declaration: `Function Now: TDateTime;`

Description: `Now` returns the current date and time. It is equivalent to `Date+Time`.

Errors: None.

See also: `Date` ([405](#)), `Time` ([417](#))

Listing: `sysutex/ex18.pp`

```
Program Example18;  
  
{ This program demonstrates the Now function }  
  
Uses sysutils;  
  
Begin  
  WriteLn ( 'Now : ', DateTimeToStr(Now));  
End.
```

StrToDate

Declaration: `Function StrToDate(const S: string): TDateTime;`

Description: `StrToDate` converts the string `S` to a `TDateTime` date value. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not supported in Delphi*)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: `StrToTime` ([416](#)), `DateToStr` ([409](#)) *n* `TimeToStr` ([418](#)).

Listing: sysutex/ex19.pp

```
Program Example19;  
  
{ This program demonstrates the StrToDate function }  
  
Uses sysutils;  
  
Procedure TestStr (S : String);  
  
begin  
  WriteLn (S, ' : ', DateToStr(StrToDate(S)));  
end;  
  
Begin  
  
  WriteLn ( 'ShortDateFormat ', ShortDateFormat );  
  TestStr(DateTimeToStr(Date));  
  TestStr('05/05/1999');  
  TestStr('5/5');  
  TestStr('5');  
End.
```

StrToDateTime

Declaration: `Function StrToDateTime(const S: string): TDateTime;`

Description: `StrToDateTime` converts the string `S` to a `TDateTime` date and time value. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not supported in Delphi*)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: `StrToDate` ([415](#)), `StrToTime` ([416](#)), `DateTimeToStr` ([406](#))

Listing: sysutex/ex20.pp

Program Example20;

{ This program demonstrates the StrToDateTime function }

Uses sysutils;

Procedure TestStr (S : **String**);

begin

WriteLn (S, ' : ', **DateTimeToStr** (**StrToDateTime**(S)));
end;

Begin

WriteLn ('ShortDateFormat ', ShortDateFormat);
 TestStr (**DateTimeToStr** (**Now**));
 TestStr ('05-05-1999 15:50');
 TestStr ('5-5 13:30');
 TestStr ('5 1:30PM');
End.

StrToTime

Declaration: `Function StrToTime(const S: string): TDateTime;`

Description: `StrToTime` converts the string `S` to a `TDateTime` time value. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: `StrToDate` ([415](#)), `StrToDateTime` ([415](#)), `TimeToStr` ([418](#))

Listing: sysutex/ex21.pp

Program Example21;

{ This program demonstrates the StrToTime function }

Uses sysutils;

Procedure TestStr (S : **String**);

begin

WriteLn (S, ' : ', **TimeToStr** (**StrToTime**(S)));
end;

Begin

teststr (**TimeToStr** (**Time**));
 teststr ('12:00');

```
    teststr ('15:30');  
    teststr ('3:30PM');  
End.
```

SystemTimeToDateTime

Declaration: Function SystemTimeToDateTime(const SystemTime: TSystemTime): TDateTime;

Description: SystemTimeToDateTime converts a TSystemTime record to a TDateTime style date/time indication.

Errors: None.

See also: DateTimeToSystemTime ([408](#))

Listing: sysutex/ex22.pp

Program Example22;

{ This program demonstrates the SystemTimeToDateTime function }

Uses sysutils;

Var ST : TSystemTime;

Begin

DateTimeToSystemTime(**Now**,ST);

With St **do**

begin

WriteLn ('Today is ',year,'/',month,'/',Day);

WriteLn ('The time is ',Hour,':',minute,':',Second,'.',MilliSecond);

end;

WriteLn ('Converted : ',DateTimeToStr(SystemTimeToDateTime(ST)));

End.

Time

Declaration: Function Time: TDateTime;

Description: Time returns the current time in TDateTime format. The date part of the TDateTimeValue is set to zero.

Errors: None.

See also: Now ([414](#)), Date ([405](#))

Listing: sysutex/ex23.pp

Program Example23;

{ This program demonstrates the Time function }

Uses sysutils;

Begin

WriteLn ('The time is : ',TimeToStr(Time));

End.

TimeStampToDateTime

Declaration: `Function TimeStampToDateTime(const TimeStamp: TTimeStamp): TDateTime;`

Description: `TimeStampToDateTime` converts `TimeStamp` to a `TDateTime` format variable. It is the inverse operation of `DateTimeToTimeStamp` (408).

Errors: None.

See also: `DateTimeToTimeStamp` (408), `TimeStampToMSecs` (418)

Listing: `sysutex/ex24.pp`

Program `Example24;`

{ This program demonstrates the TimeStampToDateTime function }

Uses `sysutils;`

Var `TS : TTimeStamp;`
`DT : TDateTime;`

Begin

`TS:=DateTimeToTimeStamp (Now);`

With `TS do`

begin

`WriteLn ('Now is ',time,' millisecond past midnight');`

`WriteLn ('Today is ' ,Date,' days past 1/1/0001');`

end;

`DT:=TimeStampToDateTime(TS);`

`WriteLn ('Together this is : ',DateTimeToStr(DT));`

End.

TimeStampToMSecs

Declaration: `Function TimeStampToMSecs(const TimeStamp: TTimeStamp): comp;`

Description: `TimeStampToMSecs` converts `TimeStamp` to the number of seconds since 1/1/0001.

Use `TTimeStamp` variables if you need to keep very precise track of time.

Errors: None.

See also: `MSecsToTimeStamp` (414), `TimeStampToDateTime` (418)

For an example, see `MSecsToTimeStamp` (414).

TimeToStr

Declaration: `Function TimeToStr(Time: TDateTime): string;`

Description: `TimeToStr` converts the time in `Time` to a string. It uses the `ShortTimeFormat` variable to see what formatting needs to be applied. It is therefor entirely equivalent to a `FormatDateTime('t',Time)` call.

Errors: None.

See also:

Listing: sysutex/ex25.pp

Program Example25;

```
{ This program demonstrates the TimeToStr function }
```

Uses sysutils;

Begin

```
  WriteLn ('The current time is : ', TimeToStr(Time));  
End.
```

22.5 Disk functions

AddDisk (Linux only)

Declaration: Function AddDisk (Const Path : String) : Longint;

Description: On Linux both the DiskFree ([45](#)) and DiskSize ([46](#)) functions need a file on the specified drive, since is required for the statfs system call.

These filenames are set in drivestr[0..26], and the first 4 have been preset to :

Disk 0 ' . ' default drive - hence current directory is used.

Disk 1 ' /fd0 / . ' floppy drive 1.

Disk 2 ' /fd1 / . ' floppy drive 2.

Disk 3 ' / ' C: equivalent of DOS is the root partition.

Drives 4..26 can be set by your own applications with the AddDisk call.

The AddDisk call adds Path to the names of drive files, and returns the number of the disk that corresponds to this drive. If you add more than 21 drives, the count is wrapped to 4.

Errors: None.

See also: DiskFree ([420](#)), DiskSize ([420](#))

CreateDir

Declaration: Function CreateDir(Const NewDir : String) : Boolean;

Description: CreateDir creates a new directory with name NewDir. If the directory doesn't contain an absolute path, then the directory is created below the current working directory.

The function returns True if the directory was successfully created, False otherwise.

Errors: In case of an error, the function returns False.

See also: RemoveDir ([421](#))

Listing: sysutex/ex26.pp

Program Example26;

```
{ This program demonstrates the CreateDir and RemoveDir functions }  
{ Run this program twice in the same directory }
```

Uses sysutils;

```
Begin
  If Not FileExists('NewDir') then
    If Not CreateDir('NewDir') Then
      Writeln('Failed to create directory !')
    else
      Writeln('Created "NewDir" directory')
  Else
    If Not RemoveDir('NewDir') Then
      Writeln('Failed to remove directory !')
    else
      Writeln('Removed "NewDir" directory');
End.
```

DiskFree

Declaration: `Function DiskFree(Drive : Byte) : Int64;`

Description: `DiskFree` returns the free space (in bytes) on disk `Drive`. `Drive` is the number of the disk drive:

- 0 for the current drive.
- 1 for the first floppy drive.
- 2 for the second floppy drive.
- 3 for the first hard-disk partition.
- 4-26 for all other drives and partitions.

Remark Under LINUX, and Unix in general, the concept of disk is different than the DOS one, since the filesystem is seen as one big directory tree. For this reason, the `DiskFree` and `DiskSize` (46) functions must be mimicked using filenames that reside on the partitions. For more information, see `AddDisk` (419)

Errors: On error, -1 is returned.

See also: `DiskSize` (420), `AddDisk` (419)

Listing: `sysutex/ex27.pp`

Program `Example27;`

{ This program demonstrates the DiskFree function }

Uses `sysutils;`

```
Begin
  Write('Size of current disk      : ', DiskSize(0));
  Writeln(' (= ', DiskSize(0) div 1024, 'k)');
  Write('Free space of current disk : ', Diskfree(0));
  Writeln(' (= ', Diskfree(0) div 1024, 'k)');
End.
```

DiskSize

Declaration: `Function DiskSize(Drive : Byte) : Int64;`

Description: `DiskSize` returns the size (in bytes) of disk `Drive`. `Drive` is the number of the disk drive:

0for the current drive.

1for the first floppy drive.

2for the second floppy drive.

3for the first hard-disk partition.

4-26for all other drives and partitions.

Remark Under LINUX, and Unix in general, the concept of disk is different than the DOS one, since the filesystem is seen as one big directory tree. For this reason, the **DiskFree** (45) and **DiskSize** functions must be mimicked using filenames that reside on the partitions. For more information, see **AddDisk** (419)

Errors: On error, -1 is returned.

See also: **DiskFree** (420), **AddDisk** (419)

For an example, see **DiskFree** (420).

GetCurrentDir

Declaration: `Function GetCurrentDir : String;`

Description: **GetCurrentDir** returns the current working directory.

Errors: None.

See also: **SetCurrentDir** (422), **DiskFree** (45), **DiskSize** (46)

Listing: sysutex/ex28.pp

Program Example28;

{ This program demonstrates the GetCurrentDir function }

Uses sysutils;

Begin

WriteLn ('Current Directory is : ',GetCurrentDir);

End.

RemoveDir

Declaration: `Function RemoveDir(Const Dir : String) : Boolean;`

Description: **RemoveDir** removes directory **Dir** from the disk. If the directory is not absolute, it is appended to the current working directory.

Errors: In case of error (e.g. the directory isn't empty) the function returns **False**. If successful, **True** is returned.

See also:

For an example, see **CreateDir** (419).

SetCurrentDir

Declaration: `Function SetCurrentDir(Const NewDir : String) : Boolean;`

Description: `SetCurrentDir` sets the current working directory of your program to `NewDir`. It returns `True` if the function was successful, `False` otherwise.

Errors: In case of error, `False` is returned.

See also: `GetCurrentDir` ([421](#))

Listing: `sysutex/ex29.pp`

Program `Example29;`

{ This program demonstrates the SetCurrentDir function }

Uses `sysutils;`

Begin

```
  If SetCurrentDir ( '..' ) Then
    WriteLn ( 'Now in directory ', GetCurrentDir)
  else
    WriteLn ( 'Change directory to .. failed.' );
```

End.

22.6 File handling functions

ChangeFileExt

Declaration: `Function ChangeFileExt(const FileName, Extension: string): string;`

Description: `ChangeFileExt` changes the file extension in `FileName` to `Extension`. The extension `Extension` includes the starting `.` (dot). The previous extension of `FileName` are all characters after the last `.`, the `.` character included.

If `FileName` doesn't have an extension, `Extension` is just appended.

Errors: None.

See also: `ExtractFileName` ([425](#)), `ExtractFilePath` ([426](#)), `ExpandFileName` ([423](#))

DeleteFile

Declaration: `Function DeleteFile(Const FileName : String) : Boolean;`

Description: `DeleteFile` deletes file `FileName` from disk. The function returns `True` if the file was successfully removed, `False` otherwise.

Errors: On error, `False` is returned.

See also: `FileCreate` ([427](#)), `FileExists` ([428](#))

Listing: `sysutex/ex31.pp`

```
Program Example31;

{ This program demonstrates the DeleteFile function }

Uses sysutils;

Var
  Line : String;
  F, I : Longint;

Begin
  F:=FileCreate('test.txt');
  Line:='Some string line.'#10;
  For I:=1 to 10 do
    FileWrite (F,Line[I],Length(Line));
  FileClose(F);
  DeleteFile('test.txt');
End.
```

DoDirSeparators

Declaration: `Procedure DoDirSeparators(Var FileName : String);`

Description: This function replaces all directory separators ' and '/' to the directory separator character for the current system.

Errors: None.

See also: [ExtractFileName \(425\)](#), [ExtractFilePath \(426\)](#)

Listing: sysutex/ex32.pp

```
Program Example32;

{ This program demonstrates the DoDirSeparators function }
{$H+}

Uses sysutils;

Procedure Testit (F : String);

begin
  Writeln ('Before : ',F);
  DoDirSeparators (F);
  Writeln ('After  : ',F);
end;

Begin
  Testit (GetCurrentDir);
  Testit ('c:\pp\bin\win32');
  Testit ('/usr/lib/fpc');
  Testit ('\usr\lib\fpc');
End.
```

ExpandFileName

Declaration: `Function ExpandFileName(Const FileName : string): String;`

Description: `ExpandFileName` expands the filename to an absolute filename. It changes all directory separator characters to the one appropriate for the system first.

Errors: None.

See also: `ExtractFileName` (425), `ExtractFilePath` (426), `ExtractFileDir` (424), `ExtractFileDrive` (425), `ExtractFileExt` (425), `ExtractRelativePath` (426)

Listing: `sysutex/ex33.pp`

Program `Example33`;

{ This program demonstrates the ExpandFileName function }

Uses `sysutils`;

Procedure `Testit (F : String)`;

begin

WriteLn (F, ' expands to : ', `ExpandFileName(F)`);
end;

Begin

`Testit('ex33.pp');`
 `Testit(ParamStr(0));`
 `Testit('/pp/bin/win32/ppc386');`
 `Testit('\pp\bin\win32\ppc386');`
 `Testit('.');`

End.

ExpandUNCFileName

Declaration: `Function ExpandUNCFileName(Const FileName : string): String;`

Description: `ExpandUNCFileName` runs `ExpandFileName` (423) on `FileName` and then attempts to replace the driveletter by the name of a shared disk.

Errors:

See also: `ExtractFileName` (425), `ExtractFilePath` (426), `ExtractFileDir` (424), `ExtractFileDrive` (425), `ExtractFileExt` (425), `ExtractRelativePath` (426)

ExtractFileDir

Declaration: `Function ExtractFileDir(Const FileName : string): string;`

Description: `ExtractFileDir` returns only the directory part of `FileName`, not including a driveletter. The directory name has NO ending directory separator, in difference with `ExtractFilePath` (426).

Errors: None.

See also: `ExtractFileName` (425), `ExtractFilePath` (426), `ExtractFileDir` (424), `ExtractFileDrive` (425), `ExtractFileExt` (425), `ExtractRelativePath` (426)

Listing: `sysutex/ex34.pp`

Program Example34;

```
{ This program demonstrates the ExtractFileName function }
{$H+}
Uses sysutils;
```

```
Procedure Testit(F : String);
```

```
begin
  Writeln ( 'FileName      : ', F);
  Writeln ( 'Has Name     : ', ExtractFileName(F));
  Writeln ( 'Has Path     : ', ExtractFilePath(F));
  Writeln ( 'Has Extension : ', ExtractFileExt(F));
  Writeln ( 'Has Directory : ', ExtractFileDir(F));
  Writeln ( 'Has Drive    : ', ExtractFileDrive(F));
end;
```

```
Begin
  Testit ( Paramstr(0));
  Testit ( '/usr/local/bin/mysql' );
  Testit ( 'c:\pp\bin\win32\ppc386.exe' );
  Testit ( '/pp/bin/win32/ppc386.exe' );
End.
```

ExtractFileDrive

Declaration: `Function ExtractFileDrive(const FileName: string): string;`

Description: `Extract`

Errors:

See also: [ExtractFileName \(425\)](#), [ExtractFilePath \(426\)](#), [ExtractFileDir \(424\)](#), [ExtractFileDrive \(425\)](#), [ExtractFileExt \(425\)](#), [ExtractRelativePath \(426\)](#)

For an example, see [ExtractFileDir \(424\)](#).

ExtractFileExt

Declaration: `Function ExtractFileExt(const FileName: string): string;`

Description: `ExtractFileExt` returns the extension (including the `.` (dot) character) of `FileName`.

Errors: None.

See also: [ExtractFileName \(425\)](#), [ExtractFilePath \(426\)](#), [ExtractFileDir \(424\)](#), [ExtractFileDrive \(425\)](#), [ExtractFileExt \(425\)](#), [ExtractRelativePath \(426\)](#)

For an example, see [ExtractFileDir \(424\)](#).

ExtractFileName

Declaration: `Function ExtractFileName(const FileName: string): string;`

Description: `ExtractFileName` returns the filename part from `FileName`. The filename consists of all characters after the last directory separator character ('/' or '\') or drive letter.

The full filename can always be reconstructed by concatenating the result of `ExtractFilePath` (426) and `ExtractFileName`.

Errors: None.

See also: `ExtractFileName` (425), `ExtractFilePath` (426), `ExtractFileDir` (424), `ExtractFileDrive` (425), `ExtractFileExt` (425), `ExtractRelativePath` (426)

For an example, see `ExtractFileDir` (424).

ExtractFilePath

Declaration: `Function ExtractFilePath(const FileName: string): string;`

Description: `ExtractFilePath` returns the path part (including driveletter) from `FileName`. The path consists of all characters before the last directory separator character ('/' or '\'), including the directory separator itself. In case there is only a drive letter, that will be returned.

The full filename can always be reconstructed by concatenating the result of `ExtractFilePath` and `ExtractFileName` (425).

Errors: None.

See also: `ExtractFileName` (425), `ExtractFilePath` (426), `ExtractFileDir` (424), `ExtractFileDrive` (425), `ExtractFileExt` (425), `ExtractRelativePath` (426)

For an example, see `ExtractFileDir` (424).

ExtractRelativePath

Declaration: `Function ExtractRelativePath(Const BaseName, DestName : String): String;`

Description: `ExtractRelativePath` constructs a relative path to go from `BaseName` to `DestName`. If `DestName` is on another drive (Not on Linux) then the whole `Destname` is returned.

Note: This function does not exist in the Delphi unit.

Errors: None.

See also: `ExtractFileName` (425), `ExtractFilePath` (426), `ExtractFileDir` (424), `ExtractFileDrive` (425), `ExtractFileExt` (425),

Listing: sysutex/ex35.pp

Program Example35;

{ This program demonstrates the ExtractRelativePath function }

Uses sysutils;

Procedure Testit (FromDir, ToDir : **String**);

begin

Write ('From "', FromDir, '" to "', ToDir, '" via " ');

WriteLn (ExtractRelativePath (FromDir, ToDir), ' ');

end;

Begin

```
Testit ('/pp/src/compiler', '/pp/bin/win32/ppc386');  
Testit ('/pp/bin/win32/ppc386', '/pp/src/compiler');  
Testit ('e:/pp/bin/win32/ppc386', 'd:/pp/src/compiler');  
Testit ('e:\pp\bin\win32\ppc386', 'd:\pp\src\compiler');
```

End.

FileAge

Declaration: `Function FileAge(Const FileName : String): Longint;`

Description: `FileAge` returns the last modification time of file `FileName`. The `FileDate` format can be transformed to `TDateTime` format with the `FileDateToDateTime` ([411](#)) function.

Errors: In case of errors, -1 is returned.

See also: `FileDateToDateTime` ([411](#)), `FileExists` ([428](#)), `FileGetAttr` ([429](#))

Listing: `sysutex/ex36.pp`

Program `Example36;`

{ This program demonstrates the FileAge function }

Uses `sysutils;`

Var `S : TDateTime;`
 `fa : Longint;`

Begin

```
fa:=FileAge('ex36.pp');  
If Fa<>-1 then  
begin  
S:=FileDateToDateTime(fa);  
WriteLn('I'm from ', DateTimeToStr(S))  
end;
```

End.

FileClose

Declaration: `Procedure FileClose(Handle : Longint);`

Description: `FileClose` closes the file handle `Handle`. After this call, attempting to read or write from the handle will result in an error.

Errors: None.

See also: `FileCreate` ([427](#)), `FileWrite` ([433](#)), `FileOpen` ([430](#)), `FileRead` ([431](#)), `FileTruncate` ([433](#)), `FileSeek` ([432](#))

For an example, see `FileCreate` ([427](#))

FileCreate

Declaration: `Function FileCreate(Const FileName : String) : Longint;`

Description: `FileCreate` creates a new file with name `FileName` on the disk and returns a file handle which can be used to read or write from the file with the `FileRead` (431) and `FileWrite` (433) functions.

If a file with name `FileName` already existed on the disk, it is overwritten.

Errors: If an error occurs (e.g. disk full or non-existent path), the function returns `-1`.

See also: `FileClose` (427), `FileWrite` (433), `FileOpen` (430), `FileRead` (431), `FileTruncate` (433), `FileSeek` (432)

Listing: `sysutex/ex37.pp`

Program `Example37`;

{ This program demonstrates the FileCreate function }

Uses `sysutils`;

Var `I,J,F` : `Longint`;

Begin

```
F:=FileCreate ('test.dat');
If F=-1 then
  Halt(1);
For I:=0 to 100 do
  FileWrite(F,I,SizeOf(i));
FileClose(f);
F:=FileOpen ('test.dat',fmOpenRead);
For I:=0 to 100 do
  begin
    FileRead (F,J,SizeOf(J));
    If J<>I then
      Writeln ('Mismatch at file position ',I)
    end;
  FileSeek(F,0,fsFromBeginning);
  Randomize;
  Repeat
    FileSeek(F,Random(100)*4,fsFromBeginning);
    FileRead (F,J,SizeOf(J));
    Writeln ('Random read : ',j);
  Until J>80;
  FileClose(F);
  F:=FileOpen('test.dat',fmOpenWrite);
  I:=50*SizeOf(Longint);
  If FileTruncate(F,I) then
    Writeln('Successfully truncated file to ',I,' bytes. ');
  FileClose(F);
End.
```

FileExists

Declaration: `Function FileExists(Const FileName : String) : Boolean;`

Description: `FileExists` returns `True` if a file with name `FileName` exists on the disk, `False` otherwise.

Errors: None.

See also: `FileAge` (427), `FileGetAttr` (429), `FileSetAttr` (432)

Listing: sysutex/ex38.pp

Program Example38;

{ This program demonstrates the FileExists function }

Uses sysutils;

Begin

If FileExists(ParamStr(0)) **Then**

 WriteLn ('All is well, I seem to exist.');

End.

FileGetAttr

Declaration: Function FileGetAttr(Const FileName : String) : Longint;

Description: FileGetAttr returns the attribute settings of file FileName. The attribute is a OR-ed combination of the following constants:

faReadOnlyThe file is read-only.

faHiddenThe file is hidden. (On LINUX, this means that the filename starts with a dot)

faSysFileThe file is a system file (On LINUX, this means that the file is a character, block or FIFO file).

faVolumeIdVolume Label. Not possible under LINUX.

faDirectoryFile is a directory.

faArchivefile is an archive. Not possible on LINUX.

Errors: In case of error, -1 is returned.

See also: FileSetAttr ([432](#)), FileAge ([427](#)), FileGetDate ([430](#)).

Listing: sysutex/ex40.pp

Program Example40;

{ This program demonstrates the FileGetAttr function }

Uses sysutils;

Procedure Testit (Name : String);

Var F : Longint;

Begin

 F := FileGetAttr (Name);

If F <> -1 **then**

begin

 WriteLn ('Testing : ', Name);

If (F and faReadOnly) <> 0 **then**

 WriteLn ('File is ReadOnly');

If (F and faHidden) <> 0 **then**

 WriteLn ('File is hidden');

If (F and faSysFile) <> 0 **then**

 WriteLn ('File is a system file');

If (F and faVolumeId) <> 0 **then**

```
      Writeln ('File is a disk label');
    If (F and faArchive)<>0 then
      Writeln ('File is artchive file');
    If (F and faDirectory)<>0 then
      Writeln ('File is a directory');
    end
  else
    Writeln ('Error reading attribites of ',Name);
  end;

begin
  testit ('ex40.pp');
  testit (ParamStr(0));
  testit ('.');
  testit ('/');
End.
```

FileGetDate

Declaration: Function FileGetDate(Handle : Longint) : Longint;

Description: FileGetdate returns the filetime of the opened file with filehandle Handle. It is the same as FileAge (427), with this difference that FileAge only needs the file name, while FilegetDate needs an open file handle.

Errors: On error, -1 is returned.

See also: FileAge (427)

Listing: sysutex/ex39.pp

Program Example39;

{ This program demonstrates the FileGetDate function }

Uses sysutils;

Var F,D : Longint;

Begin

F:=FileCreate('test.dat');

D:=FileGetDate(D);

Writeln ('File crerated on ',DateTimeToStr(FileDateToDateTime(D)));

FileClose(F);

DeleteFile('test.dat');

End.

FileOpen

Declaration: Function FileOpen(Const FileName : string; Mode : Integer) : Longint;

Description: FileOpen opens a file with name FileName with mode Mode. Mode can be one of the following constants:

fmOpenReadThe file is opened for reading.

fmOpenWriteThe file is opened for writing.

fmOpenReadWriteThe file is opened for reading and writing.

If the file has been successfully opened, it can be read from or written to (depending on the Mode parameter) with the [FileRead \(431\)](#) and [FileWrite](#) functions.

Remark that you cannot open a file if it doesn't exist yet, i.e. it will not be created for you. If you want to create a new file, or overwrite an old one, use the [FileCreate \(427\)](#) function.

Errors: On Error, -1 is returned.

See also: [FileClose \(427\)](#), [FileWrite \(433\)](#), [FileCreate \(427\)](#), [FileRead \(431\)](#), [FileTruncate \(433\)](#), [FileSeek \(432\)](#)

For an example, see [FileRead \(431\)](#)

FileRead

Declaration: `Function FileRead(Handle : Longint; Var Buffer; Count : longint) : Longint;`

Description: `FileRead` reads Count bytes from file-handle Handle and stores them into Buffer. Buffer must be at least Count bytes long. No checking on this is performed, so be careful not to overwrite any memory. Handle must be the result of a [FileOpen \(430\)](#) call.

Errors: On error, -1 is returned.

See also: [FileClose \(427\)](#), [FileWrite \(433\)](#), [FileCreate \(427\)](#), [FileOpen \(430\)](#), [FileTruncate \(433\)](#), [FileSeek \(432\)](#)

For an example, see [FileOpen \(430\)](#)

FileSearch

Declaration: `Function FileSearch(Const Name, DirList : String) : String;`

Description: `FileSearch` looks for the file Name in DirList, where dirlist is a list of directories, separated by semicolons or colons. It returns the full filename of the first match found.

Errors: On error, an empty string is returned.

See also: [ExpandFileName \(423\)](#), [FindFirst \(434\)](#)

Listing: `sysutex/ex41.pp`

Program Example41;

{ Program to demonstrate the FileSearch function. }

Uses Sysutils;

Const

```
{ $ifdef linux }
  FN = 'find';
  P = './bin:/usr/bin';
{$else}
  FN = 'find.exe';
  P = 'c:\dos;c:\windows;c:\windows\system;c:\windows\system32';
{$endif}
```

```
begin
  Writeln ('find is in : ',FileSearch (FN,P));
end.
```

FileSeek

Declaration: `Function FileSeek(Handle,Offset,Origin : Longint) : Longint;`

Description: `FileSeek` sets the file pointer on position `Offset`, starting from `Origin`. `Origin` can be one of the following values:

fsFromBeginning`Offset` is relative to the first byte of the file. This position is zero-based. i.e. the first byte is at offset 0.

fsFromCurrent`Offset` is relative to the current position.

fsFromEnd`Offset` is relative to the end of the file. This means that `Offset` can only be zero or negative in this case.

If successful, the function returns the new file position, relative to the beginning of the file.

Remark: The abovementioned constants do not exist in Delphi.

Errors: On error, -1 is returned.

See also: [FileClose \(427\)](#), [FileWrite \(433\)](#), [FileCreate \(427\)](#), [FileOpen \(430\)](#), [FileRead \(431\)](#), [FileTruncate \(433\)](#)

Listing: `sysutex/ex42.pp`

Program `Example42;`

{ This program demonstrates the FileSetAttr function }

Uses `sysutils;`

Begin

```
  If FileSetAttr ('ex40.pp',faReadOnly or faHidden)=0 then
    Writeln ('Successfully made file hidden and read-only.')
  else
    Writeln ('Couldn't make file hidden and read-only.');
```

End.

For an example, see [FileCreate \(427\)](#)

FileSetAttr (Not on Linux)

Declaration: `Function FileSetAttr(Const Filename : String; Attr: longint) : Longint;`

Description: `FileSetAttr` sets the attributes of `FileName` to `Attr`. If the function was successful, 0 is returned, -1 otherwise.

`Attr` can be set to an OR-ed combination of the pre-defined `faXXX` constants.

Errors: On error, -1 is returned (always on linux).

See also: [FileGetAttr \(429\)](#), [FileGetDate \(430\)](#), [FileSetDate \(433\)](#).

FileSetDate (Not on Linux)

Declaration: `Function FileSetDate(Handle, Age : Longint) : Longint;`

Description: `FileSetDate` sets the file date of the file with handle `Handle` to `Age`, where `Age` is a DOS date-and-time stamp value.

The function returns zero if successful.

Errors: On Linux, -1 is always returned, since this is impossible to implement. On Windows and DOS, a negative error code is returned.

See also:

FileTruncate

Declaration: `Function FileTruncate(Handle, Size: Longint) : boolean;`

Description: `FileTruncate` truncates the file with handle `Handle` to `Size` bytes. The file must have been opened for writing prior to this call. The function returns `True` if successful, `False` otherwise.

Errors: On error, the function returns `False`.

See also: [FileClose \(427\)](#), [FileWrite \(433\)](#), [FileCreate \(427\)](#), [FileOpen \(430\)](#), [FileRead \(431\)](#), [FileSeek \(432\)](#)

For an example, see [FileCreate \(427\)](#).

FileWrite

Declaration: `Function FileWrite(Handle : Longint; Var Buffer; Count : Longint) : Longint;`

Description: `FileWrite` writes `Count` bytes from `Buffer` to the file with handle `Handle`. Prior to this call, the file must have been opened for writing. `Buffer` must be at least `Count` bytes large, or a memory access error may occur.

The function returns the number of bytes written, or -1 in case of an error.

Errors: In case of error, -1 is returned.

See also: [FileClose \(427\)](#), [FileCreate \(427\)](#), [FileOpen \(430\)](#), [FileRead \(431\)](#), [FileTruncate \(433\)](#), [FileSeek \(432\)](#)

For an example, see [FileCreate \(427\)](#).

FindClose

Declaration: `Procedure FindClose(Var F : TSearchrec);`

Description: `FindClose` ends a series of [FindFirst \(434\)](#)/[FindNext \(434\)](#) calls, and frees any memory used by these calls. It is *absolutely* necessary to do this call, or huge memory losses may occur.

Errors: None.

See also: [FindFirst \(434\)](#), [FindNext \(434\)](#).

For an example, see [FindFirst \(434\)](#).

FindFirst

Declaration: Function FindFirst(Const Path : String; Attr : Longint; Var Rslt : TSearchRec) : Longint;

Description: FindFirst looks for files that match the name (possibly with wildcards) in Path and attributes Attr. It then fills up the Rslt record with data gathered about the file. It returns 0 if a file matching the specified criteria is found, a nonzero value (-1 on linux) otherwise.

The Rslt record can be fed to subsequent calls to FindNext, in order to find other files matching the specifications.

remark: A FindFirst call must *always* be followed by a FindClose (433) call with the same Rslt record. Failure to do so will result in memory loss.

Errors: On error the function returns -1 on linux, a nonzero error code on Windows.

See also: FindClose (49)FindCloseSys, FindNext (434).

Listing: sysutex/ex43.pp

Program Example43;

{ This program demonstrates the FindFirst function }

Uses SysUtils;

Var Info : TSearchRec;
Count : Longint;

Begin

Count:=0;

If FindFirst ('/*',faAnyFile **and** faDirectory,Info)=0 **then**

begin

Repeat

Inc(Count);

With Info **do**

begin

If (Attr **and** faDirectory) = faDirectory **then**

Write('Dir : ');

WriteLn (Name:40,Size:15);

end;

Until FindNext(info)<>0;

end;

FindClose(Info);

WriteLn ('Finished search. Found ',Count,' matches');

End.

FindNext

Declaration: Function FindNext(Var Rslt : TSearchRec) : Longint;

Description: FindNext finds a next occurrence of a search sequence initiated by FindFirst. If another record matching the criteria in Rslt is found, 0 is returned, a nonzero constant is returned otherwise.

remark: The last FindNext call must *always* be followed by a FindClose call with the same Rslt record. Failure to do so will result in memory loss.

Errors: On error (no more file is found), a nonzero constant is returned.

See also: [FindFirst \(434\)](#), [FindClose \(49\)](#)

For an example, see [FindFirst \(434\)](#)

GetDirs

Declaration: Function GetDirs(Var DirName : String; Var Dirs : Array of pchar)
: Longint;

Description: GetDirs splits DirName in a null-byte separated list of directory names, Dirs is an array of PChars, pointing to these directory names. The function returns the number of directories found, or -1 if none were found. DirName must contain only OSDirSeparator as Directory separator chars.

Errors: None.

See also: [ExtractRelativePath \(426\)](#)

Listing: sysutex/ex45.pp

Program Example45;

```
{ This program demonstrates the GetDirs function }  
{$H+}
```

Uses sysutils;

```
Var Dirs : Array[0..127] of pchar;  
    I, Count : longint;  
    Dir, NewDir : String;
```

Begin

```
Dir:=GetCurrentDir;  
WriteLn ( 'Dir : ', Dir );  
NewDir:= '' ;  
count:=GetDirs ( Dir, Dirs );  
For I:=0 to Count do  
begin  
    NewDir:=NewDir+'/' +StrPas ( Dirs [ I ] );  
    WriteLn ( NewDir );  
end;
```

End.

RenameFile

Declaration: Function RenameFile(Const OldName, NewName : String) : Boolean;

Description: RenameFile renames a file from OldName to NewName. The function returns True if successful, False otherwise.

Remark: you cannot rename across disks or partitions.

Errors: On Error, False is returned.

See also: [DeleteFile \(422\)](#)

Listing: sysutex/ex44.pp

```
Program Example44;  
  
{ This program demonstrates the RenameFile function }  
  
Uses sysutils;  
  
Var F : Longint;  
      S : String;  
  
Begin  
  S:= 'Some short file .';  
  F:= FileCreate ( ' test.dap' );  
  FileWrite(F,S[1],Length(S));  
  FileClose(F);  
  If RenameFile ( ' test.dap', ' test.dat' ) then  
    WriteLn ( ' Successfully renamed files .' );  
End.
```

SetDirSeparators

Declaration: `Function SetDirSeparators(Const FileName : String) : String;`

Description: `SetDirSeparators` returns `FileName` with all possible `DirSeparators` replaced by `OSDirSeparator`.

Errors: None.

See also: `ExpandFileName` ([423](#)), `ExtractFilePath` ([426](#)), `ExtractFileDir` ([424](#))

Listing: `sysutex/ex47.pp`

```
Program Example47;  
  
{ This program demonstrates the SetDirSeparators function }  
  
Uses sysutils;  
  
Begin  
  WriteLn ( SetDirSeparators ( ' /pp\bin\win32\ppc386' ) );  
End.
```

22.7 PChar functions

Introduction

Most PChar functions are the same as their counterparts in the `STRINGS` unit. The following functions are the same :

1. `StrCat` ([386](#)) : Concatenates two PChar strings.
2. `StrComp` ([387](#)) : Compares two PChar strings.
3. `StrCopy` ([387](#)) : Copies a PChar string.
4. `StrECopy` ([388](#)) : Copies a PChar string and returns a pointer to the terminating null byte.
5. `StrEnd` ([389](#)) : Returns a pointer to the terminating null byte.

6. **StrlComp** (389) : Case insensitive compare of 2 PChar strings.
7. **StrLCat** (390) : Appends at most L characters from one PChar to another PChar.
8. **StrLComp** (390) : Case sensitive compare of at most L characters of 2 PChar strings.
9. **StrLCopy** (391) : Copies at most L characters from one PChar to another.
10. **StrLen** (391) : Returns the length (exclusive terminating null byte) of a PChar string.
11. **StrLlComp** (392) : Case insensitive compare of at most L characters of 2 PChar strings.
12. **StrLower** (392) : Converts a PChar to all lowercase letters.
13. **StrMove** (392) : Moves one PChar to another.
14. **StrNew** (393) : Makes a copy of a PChar on the heap, and returns a pointer to this copy.
15. **StrPos** (394) : Returns the position of one PChar string in another?
16. **StrRScan** (395) : returns a pointer to the last occurrence of on PChar string in another one.
17. **StrScan** (395) : returns a pointer to the first occurrence of on PChar string in another one.
18. **StrUpper** (395) : Converts a PChar to all uppercase letters.

The subsequent functions are different from their counterparts in **STRINGS**, although the same examples can be used.

StrAlloc

Declaration: `Function StrAlloc(Size: cardinal): PChar;`

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating #0 included, and returns a pointer to it.

Additionally, `StrAlloc` allocates 4 extra bytes to store the size of the allocated memory. Therefore this function is NOT compatible with the `StrAlloc` (386) function of the `Strings` unit.

Errors: None.

See also: `StrBufSize` (437), `StrDispose` (438), `StrAlloc` (386)

For an example, see `StrBufSize` (437).

StrBufSize

Declaration: `Function StrBufSize(var Str: PChar): cardinal;`

Description: `StrBufSize` returns the memory allocated for `Str`. This function ONLY gives the correct result if `Str` was allocated using `StrAlloc` (437).

Errors: If no more memory is available, a runtime error occurs.

See also: `StrAlloc` (437).`StrDispose` (438).

Listing: `sysutex/ex46.pp`

Program Example46;

```
{ This program demonstrates the StrBufSize function }  
{ $H+ }
```

Uses sysutils;

Const S = 'Some nice string';

Var P : Pchar;

Begin

```
P:= StrAlloc (Length(S)+1);  
StrPCopy(P,S);  
Write (P, ' has length ',length(S));  
Writeln ( ' and buffer size ', StrBufSize(P));  
StrDispose(P);
```

End.

StrDispose

Declaration: Procedure StrDispose(var Str: PChar);

Description: StrDispose frees any memory allocated for Str. This function will only function correctly if Str has been allocated using StrAlloc (437) from the SYSUTILS unit.

Errors: If an invalid pointer is passed, or a pointer not allocated with StrAlloc, an error may occur.

See also: StrBufSize (437), StrAlloc (437), StrDispose (388)

For an example, see StrBufSize (437).

StrPCopy

Declaration: Function StrPCopy(Dest: PChar; Source: string): PChar;

Description: StrPCopy Converts the Ansistring in Source to a Null-terminated string, and copies it to Dest. Dest needs enough room to contain the string Source, i.e. Length(Source)+1 bytes.

Errors: No checking is performed to see whether Dest points to enough memory to contain Source.

See also: StrPLCopy (438), StrPCopy (394)

For an example, see StrPCopy (394).

StrPLCopy

Declaration: Function StrPLCopy(Dest: PChar; Source: string; MaxLen: cardinal): PChar;

Description: StrPLCopy Converts maximally MaxLen characters of the Ansistring in Source to a Null-terminated string, and copies it to Dest. Dest needs enough room to contain the characters.

Errors: No checking is performed to see whether Dest points to enough memory to contain L characters of Source.

Errors:

See also: `StrPCopy` ([438](#)).

StrPas

Declaration: `Function StrPas(Str: PChar): string;`

Description: Converts a null terminated string in `Str` to an Ansistring, and returns this string. This string is NOT truncated at 255 characters as is the

Errors: None.

See also: `StrPas` ([393](#)).

For an example, see `StrPas` ([393](#)).

22.8 String handling functions

AdjustLineBreaks

Declaration: `Function AdjustLineBreaks(const S: string): string;`

Description: `AdjustLineBreaks` will change all `#13` characters with `#13#10` on WINDOWS NT and DOS. On LINUX, all `#13#10` character pairs are converted to `#10` and single `#13` characters also.

Errors: None.

See also: `AnsiCompareStr` ([439](#)), `AnsiCompareText` ([440](#))

Listing: `sysutex/ex48.pp`

Program `Example48;`

{ This program demonstrates the AdjustLineBreaks function }

Uses `sysutils;`

Const

`S = 'This is a string'#13'with embedded'#10'linefeed and'+
#13'CR characters';`

Begin

`WriteLn (AdjustLineBreaks(S));`

End.

AnsiCompareStr

Declaration: `Function AnsiCompareStr(const S1, S2: string): integer;`

Description: `AnsiCompareStr` compares two strings and returns the following result:

`<0`if `S1<S2`.

`0`if `S1=S2`.

`>0`if `S1>S2`.

the comparison takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to `AnsiCompareText` (440), the comparison is case sensitive.

Errors: None.

See also: `AdjustLineBreaks` (439), `AnsiCompareText` (440)

Listing: sysutex/ex49.pp

Program Example49;

```
{ This program demonstrates the AnsiCompareStr function }  
{ $H+ }
```

Uses sysutils;

Procedure TestIt (S1,S2 : **String**);

Var R : Longint;

begin

 R:=**AnsiCompareStr**(S1,S2);

Write (''' ,S1, ' is ');

If R<0 **then**

write ('less than ')

else If R=0 **then**

Write ('equal to ')

else

Write ('larger than ');

Writeln (''' ,S2, ''');

end;

Begin

 TestIt('One string','One smaller string');

 TestIt('One string','one string');

 TestIt('One string','One string');

 TestIt('One string','One tall string');

End.

AnsiCompareText

Declaration: `Function AnsiCompareText(const S1, S2: string): integer;`

Description:

Description: `AnsiCompareText` compares two strings and returns the following result:

<0 if S1<S2.

0 if S1=S2.

>0 if S1>S2.

the comparison takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to `AnsiCompareStr` (439), the comparison is case insensitive.

Errors: None.

See also: `AdjustLineBreaks` (439), `AnsiCompareText` (440)

Listing: sysutex/ex50.pp

Program Example49;

```
{ This program demonstrates the AnsiCompareText function }
{$H+}
```

Uses sysutils;

Procedure TestIt (S1,S2 : **String**);

Var R : Longint;

begin

 R:=AnsiCompareText(S1,S2);

Write ('',S1, ' is ');

If R<0 **then**

write ('less than ')

else If R=0 **then**

Write ('equal to ')

else

Write ('larger than ');

Writeln ('',S2, '');

end;

Begin

 Testit('One string','One smaller string');

 Testit('One string','one string');

 Testit('One string','One string');

 Testit('One string','One tall string');

End.

AnsiExtractQuotedStr

Declaration: Function AnsiExtractQuotedStr(var Src: PChar; Quote: Char): string;

Description: AnsiExtractQuotedStr Returns Src as a string, with Quote characters removed from the beginning and end of the string, and double Quote characters replaced by a single Quote characters. As such, it reverses the action of AnsiQuotedStr ([443](#)).

Errors: None.

See also: AnsiQuotedStr ([443](#))

Listing: sysutex/ex51.pp

Program Example51;

```
{ This program demonstrates the AnsiQuotedStr function }
```

Uses sysutils;

Var S : AnsiString;

Begin

 S:='He said "Hello" and walked on';

 S:=AnsiQuotedStr(Pchar(S), ' ');

Writeln (S);

```
    WriteLn (AnsiExtractQuotedStr (Pchar(S), ''' ));  
End.
```

AnsiLastChar

Declaration: `Function AnsiLastChar(const S: string): PChar;`

Description: This function returns a pointer to the last character of S. Since multibyte characters are not yet supported, this is the same as `@S[Length(S)]`.

Errors: None.

See also: `AnsiStrLastChar` ([445](#))

Listing: sysutex/ex52.pp

Program Example52;

{ This program demonstrates the AnsiLastChar function }

Uses sysutils;

Var S : AnsiString;
 L : Longint;

Begin

```
    S:='This is an ansistring.';  
    WriteLn ( 'Last character of S is : ',AnsiLastChar(S));  
    L:=Longint (AnsiLastChar(S))-Longint (@S[1])+1;  
    WriteLn ( 'Length of S is : ',L);
```

End.

AnsiLowerCase

Declaration: `Function AnsiLowerCase(const s: string): string;`

Description: `AnsiLowerCase` converts the string S to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On linux, no language setting is taken in account yet.

Errors: None.

See also: `AnsiUpperCase` ([448](#)), `AnsiStrLower` ([447](#)), `AnsiStrUpper` ([448](#))

Listing: sysutex/ex53.pp

Program Example53;

{ This program demonstrates the AnsiLowerCase function }

Uses sysutils;

Procedure Testit (S : String);

begin

```
    WriteLn (S, ' -> ', AnsiLowerCase(S))
end;
```

```
Begin
    Testit('AN UPPERCASE STRING');
    Testit('Some mixed SString');
    Testit('a lowercase string');
End.
```

AnsiQuotedStr

Declaration: `Function AnsiQuotedStr(const S: string; Quote: char): string;`

Description: `AnsiQuotedString` quotes the string `S` and returns the result. This means that it puts the `Quote` character at both the beginning and end of the string and replaces any occurrence of `Quote` in `S` with 2 `Quote` characters. The action of `AnsiQuotedString` can be reversed by `AnsiExtractQuotedStr` (441).

Errors: None.

See also: `AnsiExtractQuotedStr` (441)

For an example, see `AnsiExtractQuotedStr` (441)

AnsiStrComp

Declaration: `Function AnsiStrComp(S1, S2: PChar): integer;`

Description: `AnsiStrComp` compares 2 `PChar` strings, and returns the following result:

```
<0if S1<S2.
0if S1=S2.
>0if S1>S2.
```

The comparison of the two strings is case-sensitive. The function does not yet take internationalization settings into account.

Errors: None.

See also: `AnsiCompareText` (440), `AnsiCompareStr` (439)

Listing: `sysutex/ex54.pp`

Program `Example54;`

```
{ This program demonstrates the AnsiStrComp function }
```

Uses `sysutils;`

Procedure `TestIt (S1,S2 : PChar);`

Var `R : Longint;`

```
begin
    R:=AnsiStrComp(S1,S2);
    Write ( '"',S1,'" is ' );
```

```
    If R<0 then
        write ( 'less than ' )
    else If R=0 then
        Write ( 'equal to ' )
    else
        Write ( 'larger than ' );
    Writeln ( ''' ,S2, ''' );
end;

Begin
    Testit('One string','One smaller string');
    Testit('One string','one string');
    Testit('One string','One string');
    Testit('One string','One tall string');
End.
```

AnsiStrlComp

Declaration: Function AnsiStrlComp(S1, S2: PChar): integer;

Description: AnsiStrlComp compares 2 PChar strings, and returns the following result:

```
<0if S1<S2.
0if S1=S2.
>0if S1>S2.
```

The comparison of the two strings is case-insensitive. The function does not yet take internationalization settings into account.

Errors: None.

See also: [AnsiCompareText \(440\)](#), [AnsiCompareStr \(439\)](#)

Listing: sysutex/ex55.pp

Program Example55;

```
{ This program demonstrates the AnsiStrlComp function }
```

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

```
begin
    R:=AnsiStrlComp(S1,S2);
    Write ( ''' ,S1, '' is ' );
    If R<0 then
        write ( 'less than ' )
    else If R=0 then
        Write ( 'equal to ' )
    else
        Write ( 'larger than ' );
    Writeln ( ''' ,S2, ''' );
end;
```

```
Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

AnsiStrLastChar

Declaration: `function AnsiStrLastChar(Str: PChar): PChar;`

Declaration: `AnsiStrLastChar` returns a pointer to the last character of `Str`. Since multibyte characters are not yet supported, this is the same as `StrEnd(Str)-1`.

Errors: None.

See also: `AnsiLastChar` ([442](#))

Listing: `sysutex/ex58.pp`

Program `Example58;`

{ This program demonstrates the AnsiStrLastChar function }

Uses `sysutils;`

Var `P : PChar;`
 `L : Longint;`

```
Begin
  P:= 'This is an PChar string.';
  Writeln ('Last character of P is : ',AnsiStrLastChar(P));
  L:= Longint(AnsiStrLastChar(P))- Longint(P)+1;
  Writeln ('Length of P (' ,P, ') is : ',L);
End.
```

AnsiStrLComp

Declaration: `Function AnsiStrLComp(S1, S2: PChar; MaxLen: cardinal): integer;`

Description: `AnsiStrLComp` compares the first `Maxlen` characters of 2 `PChar` strings, `S1` and `S2`, and returns the following result:

<0 if `S1<S2`.

0 if `S1=S2`.

>0 if `S1>S2`.

The comparison of the two strings is case-sensitive. The function does not yet take internationalization settings into account.

Errors: None.

See also: `AnsiCompareText` ([440](#)), `AnsiCompareStr` ([439](#))

Listing: `sysutex/ex56.pp`

```
Program Example56;

{ This program demonstrates the AnsiStrLComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin
  R:=AnsiStrLComp(S1,S2,L);
  Write ( 'First ',L,' characters of "',S1,'" are ');
  If R<0 then
    write ( 'less than ' )
  else If R=0 then
    Write ( 'equal to ' )
  else
    Write ( 'larger than ' );
  WriteLn ( 'those of "',S2,'" );
end;

Begin
  TestIt('One string','One smaller string',255);
  TestIt('One string','One String',4);
  TestIt('One string','1 string',0);
  TestIt('One string','One string.',9);
End.
```

AnsiStrLIComp

Declaration: Function AnsiStrLIComp(S1, S2: PChar; MaxLen: cardinal): integer;

Description: AnsiStrLIComp compares the first MaxLen characters of 2 PChar strings, S1 and S2, and returns the following result:

<0if S1<S2.

0if S1=S2.

>0if S1>S2.

The comparison of the two strings is case-insensitive. The function does not yet take internationalization settings into account.

Errors: None.

See also: [AnsiCompareText \(440\)](#), [AnsiCompareStr \(439\)](#)

Listing: sysutex/ex57.pp

```
Program Example57;

{ This program demonstrates the AnsiStrLIComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);
```



```
Var R : Longint;

begin
  R:=AnsiStrLIComp(S1,S2,L);
  Write ( ' First ',L,' characters of "',S1,'" are ');
  If R<0 then
    write ( 'less than ')
  else If R=0 then
    Write ( 'equal to ')
  else
    Write ( 'larger than ');
  Writeln ( 'those of "',S2,'"');
end;

Begin
  Testit('One string','One smaller string',255);
  Testit('ONE STRING','one String',4);
  Testit('One string','1 STRING',0);
  Testit('One STRING','one string.',9);
End.
```

AnsiStrLower

Declaration: Function AnsiStrLower(Str: PChar): PChar;

Description: AnsiStrLower converts the PChar Str to lowercase characters and returns the resulting pchar. Note that Str itself is modified, not a copy, as in the case of AnsiLowerCase (442). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On linux, no language setting is taken in account yet.

Errors: None.

See also: AnsiStrUpper (448), AnsiLowerCase (442)

Listing: sysutex/ex59.pp

Program Example59;

{ This program demonstrates the AnsiStrLower function }

Uses sysutils;

Procedure Testit (S : Pchar);

```
begin
  Writeln (S, ' -> ',AnsiStrLower(S))
end;
```

```
Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

AnsiStrUpper

Declaration: `Function AnsiStrUpper(Str: PChar): PChar;`

Description: `AnsiStrUpper` converts the `PChar` `Str` to uppercase characters and returns the resulting string. Note that `Str` itself is modified, not a copy, as in the case of `AnsiUpperCase` (448). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On linux, no language setting is taken in account yet.

Errors: None.

See also: `AnsiUpperCase` (448), `AnsiStrLower` (447), `AnsiLowerCase` (442)

Listing: `sysutex/ex60.pp`

Program `Example60;`

{ This program demonstrates the AnsiStrUpper function }

Uses `sysutils;`

Procedure `Testit (S : Pchar);`

begin

`WriteLn (S, ' -> ', AnsiStrUpper(S))`

end;

Begin

`Testit('AN UPPERCASE STRING');`

`Testit('Some mixed STring');`

`Testit('a lowercase string');`

End.

AnsiUpperCase

Declaration: `Function AnsiUpperCase(const s: string): string;`

Description: `AnsiUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On linux, no language setting is taken in account yet.

Errors: None.

See also: `AnsiStrUpper` (448), `AnsiStrLower` (447), `AnsiLowerCase` (442)

Listing: `sysutex/ex61.pp`

Program `Example60;`

{ This program demonstrates the AnsiUpperCase function }

Uses `sysutils;`

Procedure `Testit (S : String);`

begin

```
WriteLn (S, ' -> ', AnsiUpperCase(S))
end;
```

```
Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed SString');
  Testit('a lowercase string');
End.
```

AppendStr

Declaration: `Procedure AppendStr(var Dest: String; const S: string);`

Description: `AppendStr` appends `S` to `Dest`.

This function is provided for Delphi compatibility only, since it is completely equivalent to `Dest := Dest+S`.

Errors: None.

See also: `AssignStr` ([449](#)), `NewStr` ([314](#)), `DisposeStr` ([315](#))

Listing: `sysutex/ex62.pp`

Program `Example62;`

```
{ This program demonstrates the AppendStr function }
```

```
Uses sysutils;
```

```
Var S : AnsiString;
```

```
Begin
  S:='This is an ';
  AppendStr(S, 'AnsiString');
  WriteLn ('S = "', S, '"');
End.
```

AssignStr

Declaration: `Procedure AssignStr(var P: PString; const S: string);`

Description: `AssignStr` allocates `S` to `P`. The old value of `P` is disposed of.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

Errors: None.

See also: `NewStr` ([314](#)), `AppendStr` ([449](#)), `DisposeStr` ([315](#))

Listing: `sysutex/ex63.pp`

Program `Example63;`

```
{ This program demonstrates the AssignStr function }
{$H+}
```

```
Uses sysutils;
```

```
Var P : PString;  
  
Begin  
  P:=NewStr('A first AnsiString');  
  WriteLn ( 'Before: P = "',P^,'"');  
  AssignStr(P,'A Second ansistring');  
  WriteLn ( 'After : P = "',P^,'"');  
  DisposeStr(P);  
End.
```

BCDToInt

Declaration: Function BCDToInt(Value: integer): integer;

Description: BCDToInt converts a BCD coded integer to a normal integer.

Errors: None.

See also: StrToInt ([467](#)), IntToStr ([462](#))

Listing: sysutex/ex64.pp

Program Example64;

{ This program demonstrates the BCDToInt function }

Uses sysutils;

```
Procedure Testit ( L : longint);  
begin  
  WriteLn (L, ' -> ',BCDToInt(L));  
end;
```

```
Begin  
  Testit(10);  
  Testit(100);  
  Testit(1000);  
End.
```

CompareMem

Declaration: Function CompareMem(P1, P2: Pointer; Length: cardinal): integer;

Description: CompareMem compares, byte by byte, 2 memory areas pointed to by P1 and P2, for a length of L bytes.

It returns the following values:

<0 if at some position the byte at P1 is less than the byte at the same position at P2.

0 if all L bytes are the same.

3

Errors:

See also:

CompareStr

Declaration: `Function CompareStr(const S1, S2: string): Integer;`

Description: `CompareStr` compares two strings, `S1` and `S2`, and returns the following result:

<0 if `S1`<`S2`.

0 if `S1`=`S2`.

>0 if `S1`>`S2`.

The comparison of the two strings is case-sensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: `AnsiCompareText` ([440](#)), `AnsiCompareStr` ([439](#)), `CompareText` ([451](#))

Listing: `sysutex/ex65.pp`

Program `Example65;`

```
{ This program demonstrates the CompareStr function }
{$H+}
```

Uses `sysutils;`

Procedure `TestIt (S1,S2 : String);`

Var `R : Longint;`

begin

`R:=CompareStr(S1,S2);`

`Write ('"',S1,'" is '');`

`If R<0 then`

`write ('less than ')`

`else If R=0 then`

`Write ('equal to ')`

`else`

`Write ('larger than ');`

`Writeln ('"',S2,'"');`

end;

Begin

`Testit('One string','One smaller string');`

`Testit('One string','one string');`

`Testit('One string','One string');`

`Testit('One string','One tall string');`

End.

CompareText

Declaration: `Function CompareText(const S1, S2: string): integer;`

Description: `CompareText` compares two strings, `S1` and `S2`, and returns the following result:

<0 if `S1`<`S2`.

0 if `S1`=`S2`.

>0if S1>S2.

The comparison of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: [AnsiCompareText \(440\)](#), [AnsiCompareStr \(439\)](#), [CompareStr \(451\)](#)

Listing: sysutex/ex66.pp

Program Example66;

```
{ This program demonstrates the CompareText function }
{$H+}
```

Uses sysutils;

Procedure TestIt (S1,S2 : **String**);

Var R : Longint;

begin

 R:=CompareText(S1,S2);

Write ('', S1, ' is ');

If R<0 **then**

write ('less than ')

else if R=0 **then**

Write ('equal to ')

else

Write ('larger than ');

Writeln ('', S2, '');

end;

Begin

 TestIt('One string','One smaller string');

 TestIt('One string','one string');

 TestIt('One string','One string');

 TestIt('One string','One tall string');

End.

DisposeStr

Declaration: Procedure DisposeStr(S: PString);

Description: DisposeStr removes the dynamically allocated string S from the heap, and releases the occupied memory.

This function is provided for Delphi compatibility only. AnsiStrings are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

Errors: None.

See also: [NewStr \(314\)](#), [AppendStr \(449\)](#), [AssignStr \(449\)](#)

For an example, see [DisposeStr \(315\)](#).

FloatToStr

Declaration: `Function FloatToStr(Value: Extended): String;`

Description: `FloatToStr` converts the floating point variable `Value` to a string representation. It will choose the shortest possible notation of the two following formats:

Fixed format will represent the string in fixed notation,

Decimal format will represent the string in scientific notation.

(more information on these formats can be found in `FloatToStrF` (453)) `FloatToStr` is completely equivalent to a `FloatToStrF(Value, ffGeneral, 15, 0);` call.

Errors: None.

See also: `FloatToStrF` (453)

Listing: `sysutex/ex67.pp`

Program `Example67;`

{ This program demonstrates the FloatToStr function }

Uses `sysutils;`

Procedure `Testit (Value : Extended);`

begin

`Writeln (Value, ' -> ', FloatToStr(Value));`

`Writeln (-Value, ' -> ', FloatToStr(-Value));`

end;

Begin

`Testit (0.0);`

`Testit (1.1);`

`Testit (1.1e-3);`

`Testit (1.1e-20);`

`Testit (1.1e-200);`

`Testit (1.1e+3);`

`Testit (1.1e+20);`

`Testit (1.1e+200);`

End.

FloatToStrF

Declaration: `Function FloatToStrF(Value: Extended; format: TFloatFormat; Precision, Digits: Integer): String;`

Description: `FloatToStrF` converts the floating point number value to a string representation, according to the settings of the parameters `Format`, `Precision` and `Digits`.

The meaning of the `Precision` and `Digits` parameter depends on the `Format` parameter. The format is controlled mainly by the `Format` parameter. It can have one of the following values:

ffcurrency **Money** format. `Value` is converted to a string using the global variables `CurrencyString`, `CurrencyFormat` and `NegCurrencyFormat`. The `Digits` parameter specifies the number of digits following the decimal point and should be in the range -1 to 18. If `Digits` equals -1, `CurrencyDecimals` is assumed. The `Precision` parameter is ignored.

ffExponentScientific format. Value is converted to a string using scientific notation: 1 digit before the decimal point, possibly preceded by a minus sign if Value is negative. The number of digits after the decimal point is controlled by Precision and must lie in the range 0 to 15.

ffFixedFixed point format. Value is converted to a string using fixed point notation. The result is composed of all digits of the integer part of Value, preceded by a minus sign if Value is negative. Following the integer part is DecimalSeparator and then the fractional part of Value, rounded off to Digits numbers. If the number is too large then the result will be in scientific notation.

ffGeneralGeneral number format. The argument is converted to a string using ffExponent or ffFixed format, depending on which one gives the shortest string. There will be no trailing zeroes. If Value is less than 0.00001 or if the number of decimals left of the decimal point is larger than Precision then scientific notation is used, and Digits is the minimum number of digits in the exponent. Otherwise Digits is ignored.

ffnumberIs the same as ffFixed, except that thousand separators are inserted in the result string.

Errors: None.

See also: FloatToStr ([453](#)), FloatToText ([455](#))

Listing: sysutex/ex68.pp

Program Example68;

```
{ This program demonstrates the FloatToStrF function }
```

Uses sysutils;

```
Const Fmt : Array [TFloatFormat] of string[10] =  
    ('general', 'exponent', 'fixed', 'number', 'Currency');
```

```
Procedure Testit (Value : Extended);
```

```
Var I, J : longint;  
    FF : TFloatFormat;
```

```
begin
```

```
    For I:=5 to 15 do
```

```
        For J:=1 to 4 do
```

```
            For FF:=ffgeneral to ffcurrency do
```

```
                begin
```

```
                    Write (Value, '(Prec: ', I:2, ', Dig: ', J, ', fmt: ', Fmt[ff], ') : ');
```

```
                    Writeln (FloatToStrf(Value, FF, I, J));
```

```
                    Write (-Value, '(Prec: ', I:2, ', Dig: ', J, ', fmt: ', Fmt[ff], ') : ');
```

```
                    Writeln (FloatToStrf(-Value, FF, I, J));
```

```
                end;
```

```
end;
```

```
Begin
```

```
    Testit (1.1);
```

```
    Testit (1.1E1);
```

```
    Testit (1.1E-1);
```

```
    Testit (1.1E5);
```

```
    Testit (1.1E-5);
```

```
    Testit (1.1E10);
```

```
    Testit (1.1E-10);
```

```
    Testit (1.1E15);
```

```
    Testit (1.1E-15);
```



```
    Testit (1.1E100);  
    Testit (1.1E-100);  
End.
```

FloatToText

Declaration: Function FloatToText(Buffer : Pchar;Value: Extended; Format: TFloatFormat;
Precision, Digits: Integer): Longint;

Description: FloatToText converts the floating point variable Value to a string representation and stores it in Buffer. The conversion is governed by format, Precision and Digits. more information on these parameters can be found in FloatToStrF (453). Buffer should point to enough space to hold the result. No checking on this is performed.

The result is the number of characters that was copied in Buffer.

Errors: None.

See also: FloatToStr (453), FloatToStrF (453)

Listing: sysutex/ex69.pp

Program Example68;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : **Array** [TFloatFormat] **of** **string**[10] =
('general', 'exponent', 'fixed', 'number', 'Currency');

Procedure Testit (Value : Extended);

Var I, J : longint;
FF : TFloatFormat;
S : ShortString;

begin

For I:=5 to 15 do

For J:=1 to 4 do

For FF:=ffgeneral to ffcurrency do

begin

Write (Value, '(Prec: ', I:2, ', Dig: ', J, ', fmt: ', Fmt[ff], ') : ');

SetLength(S, FloatToText (@S[1], Value, FF, I, J));

WriteLn (S);

Write (-Value, '(Prec: ', I:2, ', Dig: ', J, ', fmt: ', Fmt[ff], ') : ');

SetLength(S, FloatToText (@S[1], -Value, FF, I, J));

WriteLn (S);

end;

end;

Begin

Testit (1.1);

Testit (1.1E1);

Testit (1.1E-1);

Testit (1.1E5);

Testit (1.1E-5);

Testit (1.1E10);

```
Testit (1.1E-10);
Testit (1.1E15);
Testit (1.1E-15);
Testit (1.1E100);
Testit (1.1E-100);
End.
```

FmtStr

Declaration: Procedure (Var Res: String; Const Fmt : String; Const args: Array of const);

Description: FmtStr calls Format (456) with Fmt and Args as arguments, and stores the result in Res. For more information on how the resulting string is composed, see Format (456).

Errors: In case of error, a EConvertError exception is raised.

See also: Format (456), FormatBuf (461).

Listing: sysutex/ex70.pp

Program Example70;

{ This program demonstrates the FmtStr function }

Uses sysutils;

Var S : AnsiString;

Begin

S:= '';

FmtStr (S, 'For some nice examples of fomatting see %s.', ['Format']);

Writeln (S);

End.

Format

Declaration: Function Format(Const Fmt : String; const Args : Array of const) : String;

Description: Format replaces all placeholders in Fmt with the arguments passed in Args and returns the resulting string. A placeholder looks as follows:

'%' [Index':''] ['-'] [Width] ['.' Precision] ArgType

elements between single quotes must be typed as shown without the quotes, and elements between square brackets [] are optional. The meaning of the different elements is shown below:

'%' starts the placeholder. If you want to insert a literal % character, then you must insert two of them : %%.

Index ':' takes the Index-th element in the argument array as the element to insert.

'-' tells Format to left-align the inserted text. The default behaviour is to right-align inserted text. This can only take effect if the Width element is also specified.

Widththe inserted string must have at least have `Width` characters. If not, the inserted string will be padded with spaces. By default, the string is left-padded, resulting in a right-aligned string. This behaviour can be changed by the `'-'` character.

.' PrecisionIndicates the precision to be used when converting the argument. The exact meaning of this parameter depends on `ArgType`.

The `Index`, `Width` and `Precision` parameters can be replaced by `*`, in which case their value will be read from the next element in the `Args` array. This value must be an integer, or an `EConvertError` exception will be raised.

The argument type is determined from `ArgType`. It can have one of the following values (case insensitive):

DDecimal format. The next argument in the `Args` array should be an integer. The argument is converted to a decimal string,. If precision is specified, then the string will have at least `Precision` digits in it. If needed, the string is (left) padded with zeroes.

Escientific format. The next argument in the `Args` array should be a Floating point value. The argument is converted to a decimal string using scientific notation, using `FloatToStrF` (453), where the optional precision is used to specify the total number of decimals. (default a value of 15 is used). The exponent is formatted using maximally 3 digits.

In short, the `E` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffExponent, Precision, 3)
```

Ffixed point format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string, using fixed notation (see `FloatToStrF` (453)). `Precision` indicates the number of digits following the decimal point.

In short, the `F` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffFixed, fixed, 9999, Precision)
```

GGeneral number format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string using fixed point notation or scientific notation, depending on which gives the shortest result. `Precision` is used to determine the number of digits after the decimal point.

In short, the `G` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffGeneral, Precision, 3)
```

MCurrency format. the next argument in the `varArgs` array must be a floating point value. The argument is converted to a decimal string using currency notation. This means that fixed-point notation is used, but that the currency symbol is appended. If precision is specified, then then it overrides the `CurrencyDecimals` global variable used in the `FloatToStrF` (453)

In short, the `M` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffCurrency, 9999, Precision)
```

NNumber format. This is the same as fixed point format, except that thousand separators are inserted in the resulting string.

PPointer format. The next argument in the `Args` array must be a pointer (typed or untyped). The pointer value is converted to a string of length 8, representing the hexadecimal value of the pointer.

SString format. The next argument in the `Args` array must be a string. The argument is simply copied to the result string. If `Precision` is specified, then only `Precision` characters are copied to the result string.

Xhexadecimal format. The next argument in the `Args` array must be an integer. The argument is converted to a hexadecimal string with just enough characters to contain the value of the integer. If `Precision` is specified then the resulting hexadecimal representation will have at least `Precision` characters in it (with a maximum value of 32).

Errors: In case of error, an `EConversionError` exception is raised. Possible errors are:

- 1.Errors in the format specifiers.
- 2.The next argument is not of the type needed by a specifier.
- 3.The number of arguments is not sufficient for all format specifiers.

See also: `FormatBuf` ([461](#))

Listing: `sysutex/ex71.pp`

Program `example71;`

`{ $mode objfpc }`

`{ This program demonstrates the Format function }`

Uses `sysutils;`

Var `P : Pointer;`
`fmt,S : string;`

Procedure `TestInteger;`

begin

Try

```
Fmt:='[%d]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%%]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%10d]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
fmt:='[%0.4d]';S:=Format (fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%10.4d]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:d]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:10d]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:10.4d]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:-10d]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:-10.4d]';S:=Format (fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%-*.d]';S:=Format (fmt,[4,5,10]);writeln(Fmt:12,'=>',s);
```

except

On `E : Exception do`

begin

`Writeln ('Exception caught : ',E.Message);`

end;

end;

`writeln ('Press enter');`

`readln;`

end;

Procedure `TestHexaDecimal;`

begin

try

```
Fmt:='[%x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%10.4x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
```

```

Fmt:='[%0:x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:10.4x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:-10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%0:-10.4x]';S:=Format (Fmt,[10]);writeln(Fmt:12,'=>',s);
Fmt:='[%-*.x]';S:=Format (Fmt,[4,5,10]);writeln(Fmt:12,'=>',s);
except
  On E : Exception do
    begin
      Writeln ('Exception caught : ',E.Message);
    end;
end;
writeln ('Press enter');
readln;
end;

Procedure TestPointer;

begin
  P:=Pointer(1234567);
  try
    Fmt:='[0x%p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:-10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:-10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[%-*.p]';S:=Format (Fmt,[4,5,P]);writeln(Fmt:12,'=>',s);
  except
    On E : Exception do
      begin
        Writeln ('Exception caught : ',E.Message);
      end;
    end;
  writeln ('Press enter');
  readln;
end;

Procedure TestString;

begin
  try
    Fmt:='[%s]';S:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:-18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:18.12s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%-*.s]';s:=Format(fmt,[18,12,'This is a string']);Writeln(fmt:12,'=>',s);
  except
    On E : Exception do
      begin
        Writeln ('Exception caught : ',E.Message);
      end;
    end;
  writeln ('Press enter');
  readln;
end;

```

end;

Procedure TestExponential;

begin

Try

```
Fmt:='%e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%10e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%10.4e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:10e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:10.4e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:-10e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:-10.4e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%-*.*e';S:=Format (Fmt,[4,5,1.234]);writeln(Fmt:12,'=>',s);
```

except

On E : Exception do

begin

Writeln ('Exception caught : ',E.Message);

end;

end;

writeln ('Press enter');

readln;

end;

Procedure TestNegativeExponential;

begin

Try

```
Fmt:='%e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%10e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%10.4e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:10e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:10.4e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:-10e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:-10.4e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,'=>',s);
Fmt:='%-*.*e';S:=Format (Fmt,[4,5,-1.234]);writeln(Fmt:12,'=>',s);
```

except

On E : Exception do

begin

Writeln ('Exception caught : ',E.Message);

end;

end;

writeln ('Press enter');

readln;

end;

Procedure TestSmallExponential;

begin

Try

```
Fmt:='%e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,'=>',s);
Fmt:='%10e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,'=>',s);
Fmt:='%10.4e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:10e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,'=>',s);
Fmt:='%0:10.4e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,'=>',s);
```

```
Fmt:='%0:-10e';S:=Format (Fmt,[0.0123]);writeln(Fmt:12,'=>',s);
Fmt:='%0:-10.4e';S:=Format (fmt,[0.01234]);writeln(Fmt:12,'=>',s);
Fmt:='%-*.*e';S:=Format (fmt,[4,5,0.01234]);writeln(Fmt:12,'=>',s);
except
  On E : Exception do
    begin
      Writeln ('Exception caught : ',E.Message);
    end;
end;
writeln ('Press enter');
readln;
end;

Procedure TestSmallNegExponential;

begin
  Try
    Fmt:='%e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%10e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%10.4e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%0:e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%0:10e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%0:10.4e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%0:-10e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%0:-10.4e';S:=Format (fmt,[-0.01234]);writeln(Fmt:12,'=>',s);
    Fmt:='%-*.*e';S:=Format (fmt,[4,5,-0.01234]);writeln(Fmt:12,'=>',s);
  except
    On E : Exception do
      begin
        Writeln ('Exception caught : ',E.Message);
      end;
    end;
  writeln ('Press enter');
  readln;
end;

begin
  TestInteger;
  TestHexadecimal;
  TestPointer;
  TestExponential;
  TestNegativeExponential;
  TestSmallExponential;
  TestSmallNegExponential;
  teststring;
end.
```

FormatBuf

Declaration: Function FormatBuf(Var Buffer; BufLen : Cardinal; Const Fmt; fmtLen
: Cardinal; Const Args : Array of const) : Cardinal;

Description: Format

Errors:

See also:

Listing: sysutex/ex72.pp

Program Example72;

{ This program demonstrates the FormatBuf function }

Uses sysutils;

Var

S : ShortString;

Const

Fmt : ShortString = 'For some nice examples of fomatting see %s.';

Begin

S:= '';

SetLength(S, **FormatBuf** (S[1],255,Fmt[1], **Length**(Fmt),['Format']));

WriteLn (S);

End.

IntToHex

Declaration: Function IntToHex(Value: integer; Digits: integer): string;

Description: IntToHex converts Value to a hexadecimal string representation. The result will contain at least Digits characters. If Digits is less than the needed number of characters, the string will NOT be truncated. If Digits is larger than the needed number of characters, the result is padded with zeroes.

Errors: None.

See also: IntToStr ([462](#)), StrToInt

Listing: sysutex/ex73.pp

Program Example73;

{ This program demonstrates the IntToHex function }

Uses sysutils;

Var I : longint;

Begin

For I:=0 to 31 **do**

begin

WriteLn (IntToHex(1 shl I,8));

WriteLn (IntToHex(15 shl I,8))

end;

End.

IntToStr

Declaration: Function IntToStr(Value: integer): string;

Description: IntToStr coverts Value to it's string representation. The resulting string has only as much characters as needed to represent the value. If the value is negative a minus sign is prepended to the string.

Errors: None.

See also: [IntToHex \(462\)](#), [StrToInt \(467\)](#)

Listing: sysutex/ex74.pp

Program Example74;

{ This program demonstrates the IntToStr function }

Uses sysutils;

Var I : longint;

Begin

For I:=0 **to** 31 **do**

begin

WriteLn (IntToStr(1 shl I));

WriteLn (IntToStr(15 shl I));

end;

End.

IsValidIdent

Declaration: Function IsValidIdent(const Ident: string): boolean;

Description: IsValidIdent returns True if Ident can be used as a component name. It returns False otherwise. Ident must consist of a letter or underscore, followed by a combination of letters, numbers or underscores to be a valid identifier.

Errors: None.

See also:

Listing: sysutex/ex75.pp

Program Example75;

{ This program demonstrates the IsValidIdent function }

Uses sysutils;

Procedure Testit (S : String);

begin

Write (' ', S, ' is ');

If not IsValidIdent(S) **then**

Write ('NOT ');

WriteLn ('a valid identifier');

end;

Begin

 Testit ('_MyObj');

 Testit ('My__Obj1');

 Testit ('My_1_Obj');

 Testit ('1MyObject');

 Testit ('My@Object');

 Testit ('M123');

End.

LastDelimiter

Declaration: `Function LastDelimiter(const Delimiters, S: string): Integer;`

Description: `LastDelimiter` returns the *last* occurrence of any character in the set `Delimiters` in the string `S`.

Errors:

See also:

Listing: `sysutex/ex88.pp`

Program `example88;`

{ This program demonstrates the LastDelimiter function }

uses `SysUtils;`

begin

WriteLn (`LastDelimiter('\.:', 'c:\filename.ext')`);

end.

LeftStr

Declaration: `Function LeftStr(const S: string; Count: integer): string;`

Description: `LeftStr` returns the `Count` leftmost characters of `S`. It is equivalent to a call to `Copy(S, 1, Count)`.

Errors: None.

See also: `RightStr` ([466](#)), `TrimLeft` ([469](#)), `TrimRight` ([470](#)), `Trim` ([468](#))

Listing: `sysutex/ex76.pp`

Program `Example76;`

{ This program demonstrates the LeftStr function }

Uses `sysutils;`

Begin

WriteLn (`LeftStr('abcdefghijklmnopqrstuvwxyz', 20)`);

WriteLn (`LeftStr('abcdefghijklmnopqrstuvwxyz', 15)`);

WriteLn (`LeftStr('abcdefghijklmnopqrstuvwxyz', 1)`);

WriteLn (`LeftStr('abcdefghijklmnopqrstuvwxyz', 200)`);

End.

LoadStr

Declaration: `Function LoadStr(Ident: integer): string;`

Description: This function is not yet implemented. resources are not yet supported.

Errors:

See also:

LowerCase

Declaration: `Function LowerCase(const s: string): string;`

Description: `LowerCase` returns the lowercase equivalent of `S`. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the lowercase function of the system unit, and is provided for compatibility only.

Errors: None.

See also: `AnsiLowerCase` ([442](#)), `UpperCase` ([470](#)), `AnsiUpperCase` ([448](#))

Listing: `sysutex/ex77.pp`

Program `Example77;`

{ This program demonstrates the LowerCase function }

Uses `sysutils;`

Begin

`WriteLn (LowerCase('THIS WILL COME out all LoWeRcAsE !'));`

End.

NewStr

Declaration: `Function NewStr(const S: string): PString;`

Description: `NewStr` assigns a new dynamic string on the heap, copies `S` into it, and returns a pointer to the newly assigned string.

This function is obsolete, and shouldn't be used any more. The `AnsiString` mechanism also allocates anistrings on the heap, and should be preferred over this mechanism.

Errors: If not enough memory is present, an `EOutOfMemory` exception will be raised.

See also: `AssignStr` ([449](#)), `DisposeStr` ([452](#))

For an example, see `AssignStr` ([449](#)).

QuotedStr

Declaration: `Function QuotedStr(const S: string): string;`

Description: `QuotedStr` returns the string `S`, quoted with single quotes. This means that `S` is enclosed in single quotes, and every single quote in `S` is doubled. It is equivalent to a call to `AnsiQuotedStr(s, '"')`.

Errors: None.

See also: `AnsiQuotedStr` ([443](#)), `AnsiExtractQuotedStr` ([441](#)).

Listing: `sysutex/ex78.pp`

Program `Example78;`

{ This program demonstrates the QuotedStr function }

Uses sysutils;

Var S : AnsiString;

Begin

S:= 'He said ' 'Hello ' ' and walked on';

Writeln (S);

Writeln (' becomes');

Writeln (QuotedStr(S));

End.

RightStr

Declaration: Function RightStr(const S: string; Count: integer): string;

Description: RightStr returns the Count rightmost characters of S. It is equivalent to a call to Copy(S, Length(S)+1-Count, Count).

If Count is larger than the actual length of S only the real length will be used.

Errors: None.

See also: LeftStr ([464](#)), Trim ([468](#)), TrimLeft ([469](#)), TrimRight ([470](#))

Listing: sysutex/ex79.pp

Program Example79;

{ This program demonstrates the RightStr function }

Uses sysutils;

Begin

Writeln (RightStr(' abcdefghijklmnopqrstuvwxyz ',20));

Writeln (RightStr(' abcdefghijklmnopqrstuvwxyz ',15));

Writeln (RightStr(' abcdefghijklmnopqrstuvwxyz ',1));

Writeln (RightStr(' abcdefghijklmnopqrstuvwxyz ',200));

End.

StrFmt

Declaration: Function StrFmt(Buffer,Fmt : PChar; Const args: Array of const) : PChar;

Description: StrFmt will format fmt with Args, as the Format ([456](#)) function does, and it will store the result in Buffer. The function returns Buffer. Buffer should point to enough space to contain the whole result.

Errors: for a list of errors, see Format ([456](#)).

See also: StrLFmt ([467](#)), FmtStr ([456](#)), Format ([456](#)), FormatBuf ([461](#))

Listing: sysutex/ex80.pp

Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin

SetLength(S,80);

WriteLn (**StrFmt** (@S[1], 'For some nice examples of fomatting see %s.', ['Format']));
End.

StrLFmt

Declaration: Function StrLFmt(Buffer : PChar; Maxlen : Cardinal;Fmt : PChar; Const args: Array of const) : Pchar;

Description: StrLFmt will format fmt with Args, as the [Format \(456\)](#) function does, and it will store maximally Maxlen characters of the result in Buffer. The function returns Buffer. Buffer should point to enough space to contain MaxLen characters.

Errors: for a list of errors, see [Format \(456\)](#).

See also: [StrFmt \(466\)](#), [FmtStr \(456\)](#), [Format \(456\)](#), [FormatBuf \(461\)](#)

Listing: sysutex/ex81.pp

Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin

SetLength(S,80);

WriteLn (**StrLFmt** (@S[1],80,'For some nice examples of fomatting see %s.', ['Format']));
End.

StrToInt

Declaration: Function StrToInt(const s: string): integer;

Description: StrToInt will convert the string Sto an integer. If the string contains invalid characters or has an invalid format, then an EConvertError is raised.

To be successfully converted, a string can contain a combination of numerical characters, possibly preceded by a minus sign (-). Spaces are not allowed.

Errors: In case of error, an EConvertError is raised.

See also: [IntToStr \(462\)](#), [StrToIntDef \(468\)](#)

Listing: sysutex/ex82.pp

Program Example82;

{ \$mode objfpc }

```
{ This program demonstrates the StrToInt function }
```

```
Uses sysutils;
```

```
Begin
```

```
  Writeln ( StrToInt('1234'));
```

```
  Writeln ( StrToInt('-1234'));
```

```
  Writeln ( StrToInt('0'));
```

```
  Try
```

```
    Writeln ( StrToInt('12345678901234567890'));
```

```
  except
```

```
    On E : EConvertError do
```

```
      Writeln ('Invalid number encountered');
```

```
  end;
```

```
End.
```

StrToIntDef

Declaration: `Function StrToIntDef(const S: string; Default: integer): integer;`

Description: `StrToIntDef` will convert a string to an integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of numerical characters, possibly preceded by a minus sign (-). Spaces are not allowed.

Errors: None.

See also: `IntToStr` ([462](#)), `StrToInt` ([467](#))

Listing: `sysutex/ex83.pp`

Program `Example82;`

```
{ $mode objfpc }
```

```
{ This program demonstrates the StrToInt function }
```

```
Uses sysutils;
```

```
Begin
```

```
  Writeln ( StrToIntDef('1234',0));
```

```
  Writeln ( StrToIntDef('-1234',0));
```

```
  Writeln ( StrToIntDef('0',0));
```

```
  Try
```

```
    Writeln ( StrToIntDef('12345678901234567890',0));
```

```
  except
```

```
    On E : EConvertError do
```

```
      Writeln ('Invalid number encountered');
```

```
  end;
```

```
End.
```

Trim

Declaration: `Function Trim(const S: string): string;`

Description: `Trim` strips blank characters (spaces) at the beginning and end of `S` and returns the resulting string.

Only #32 characters are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: `TrimLeft` ([469](#)), `TrimRight` ([470](#))

Listing: `sysutex/ex84.pp`

Program `Example84`;

{ This program demonstrates the Trim function }

Uses `sysutils`;
`{ $H+ }`

Procedure `Testit (S : String)`;

begin
 WriteLn ('', Trim(S), '');
end;

Begin
 `Testit (' ha ha what gets lost ? ');`
 `Testit (#10#13'haha ');`
 `Testit (' ');`

End.

TrimLeft

Declaration: `Function TrimLeft(const S: string): string;`

Description: `TrimLeft` strips blank characters (spaces) at the beginning of `S` and returns the resulting string.

Only #32 characters are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: `Trim` ([468](#)), `TrimRight` ([470](#))

Listing: `sysutex/ex85.pp`

Program `Example85`;

{ This program demonstrates the TrimLeft function }

Uses `sysutils`;
`{ $H+ }`

Procedure `Testit (S : String)`;

begin
 WriteLn ('', TrimLeft(S), '');
end;

Begin

```
Testit ( '  ha ha what gets lost ? ' );  
Testit (#10#13'haha ' );  
Testit ( '          ' );  
End.
```

TrimRight

Declaration: `Function TrimRight(const S: string): string;`

Description: Trim strips blank characters (spaces) at the end of S and returns the resulting string. Only #32 characters are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: Trim ([468](#)), TrimLeft ([469](#))

Listing: sysutex/ex86.pp

Program Example86;

{ This program demonstrates the TrimRight function }

Uses sysutils;
{ \$H+ }

Procedure Testit (S : String);

begin
 WriteLn ('', TrimRight(S), '');
end;

Begin
 Testit (' ha ha what gets lost ? ');
 Testit (#10#13'haha ');
 Testit (' ');
End.

UpperCase

Declaration: `Function UpperCase(const s: string): string;`

Description: UpperCase returns the uppercase equivalent of S. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the UpCase function of the system unit, and is provided for compatibility only.

Errors: None.

See also: AnsiLowerCase ([442](#)), LowerCase ([465](#)), AnsiUpperCase ([448](#))

Errors:

See also:

Listing: sysutex/ex87.pp

Program Example87;

{ This program demonstrates the UpperCase function }

Uses sysutils;

Begin

WriteLn (**UpperCase**('this will come OUT ALL uPpErCaSe !'));

End.

Chapter 23

The TYPINFO unit

The `TypeInfo` unit contains many routines which can be used for the querying of the Run-Time Type Information (RTTI) which is generated by the compiler for classes that are compiled under the `{M+}` switch. This information can be used to retrieve or set property values for published properties for totally unknown classes. In particular, it can be used to stream classes. The `TPersistent` class in the `Classes` unit is compiled in the `{M+}` state and serves as the base class for all classes that need to be streamed.

The unit should be compatible to the Delphi 5 unit with the same name. The only calls that are still missing are the Variant calls, since Free Pascal does not support the variant type yet.

The examples in this chapter use a `rttiobj` file, which contains an object that has a published property of all supported types. It also contains some auxiliary routines and definitions.

23.1 Constants, Types and variables

Constants

The following constants are used in the implementation section of the unit.

```
BooleanIdents: array[Boolean] of String = ('False', 'True');  
DotSep: String = '.';
```

The following constants determine the access method for the `Stored` identifier of a property as used in the `PropProcs` field of the `TPropInfo` record:

```
ptField = 0;  
ptStatic = 1;  
ptVirtual = 2;  
ptConst = 3;
```

The following typed constants are used for easy selection of property types.

```
tkAny = [Low(TTypeKind)..High(TTypeKind)];  
tkMethods = [tkMethod];  
tkProperties = tkAny - tkMethods - [tkUnknown];
```

types

The following pointer types are defined:

```
PShortString = ^ShortString;  
PByte        = ^Byte;  
PWord        = ^Word;  
PLongint     = ^Longint;  
PBoolean     = ^Boolean;  
PSingle      = ^Single;  
PDouble      = ^Double;  
PExtended    = ^Extended;  
PComp        = ^Comp;  
PFixed16     = ^Fixed16;  
Variant      = Pointer;
```

The TTypeKind determines the type of a property:

```
TTypeKind = (tkUnknown, tkInteger, tkChar, tkEnumeration,  
             tkFloat, tkSet, tkMethod, tkSString, tkLString, tkAString,  
             tkWString, tkVariant, tkArray, tkRecord, tkInterface,  
             tkClass, tkObject, tkWChar, tkBool, tkInt64, tkQWord,  
             tkDynArray, tkInterfaceRaw);  
tkString = tkSString;
```

tkString is an alias that is introduced for Delphi compatibility.

If the property is an ordinal type, then TOrdType determines the size and sign of the ordinal type:

```
TOrdType = (otSByte, otUByte, otSWord, otUWord, otSLong, otULong);
```

The size of a float type is determined by TFloatType:

```
TFloatType = (ftSingle, ftDouble, ftExtended, ftComp, ftCurr,  
             ftFixed16, ftFixed32);
```

A method property (e.g. an event) can have one of several types:

```
TMethodKind = (mkProcedure, mkFunction, mkConstructor, mkDestructor,  
              mkClassProcedure, mkClassFunction);
```

The kind of parameter to a method is determined by TParamFlags:

```
TParamFlags = set of (pfVar, pfConst, pfArray, pfAddress, pfReference, pfOut);
```

Interfaces are described further with TntfFlags:

```
TIntfFlags = set of (ifHasGuid, ifDispInterface, ifDispatch);
```

The following defines a set of TTypeKind:

```
TTypeKinds = set of TTypeKind;
```

The TypeInfo function returns a pointer to a TTypeInfo record:

```
TTypeInfo = record  
  Kind : TTypeKind;  
  Name : ShortString;  
end;  
PTypeInfo = ^TTypeInfo;  
PPTypeInfo = ^PTypeInfo;
```

Note that the Name is stored with as much bytes as needed to store the name, it is not padded to 255 characters. The type data immediately follows the TTypeInfo record as a TTypeData record:

```

PTypeData = ^TTypeData;
TTypeData = packed record
case TTypeKind of
  tkUnknown,tkLString,tkWString,tkAString,tkVariant:
    ();
  tkInteger,tkChar,tkEnumeration,tkWChar:
    (OrdType : TTOrdType;
     case TTypeKind of
       tkInteger,tkChar,tkEnumeration,tkBool,tkWChar : (
         MinValue,MaxValue : Longint;
         case TTypeKind of
           tkEnumeration: (
             BaseType : PTypeInfo;
             NameList : ShortString
           )
         );
       tkSet: (
         CompType : PTypeInfo
       )
     );
  tkFloat: (
    FloatType : TFloatType
  );
  tkSString:
    (MaxLength : Byte);
  tkClass:
    (ClassType : TClass;
     ParentInfo : PTypeInfo;
     PropCount : SmallInt;
     UnitName : ShortString
    );
  tkMethod:
    (MethodKind : TMethodKind;
     ParamCount : Byte;
     ParamList : array[0..1023] of Char
     {in reality ParamList is a array[1..ParamCount] of:
    record
      Flags : TParamFlags;
      ParamName : ShortString;
      TypeName : ShortString;
    end;
    followed by
      ResultType : ShortString}
    );
  tkInt64:
    (MinInt64Value, MaxInt64Value: Int64);
  tkQWord:
    (MinQWordValue, MaxQWordValue: QWord);
  tkInterface:
    ();
end;
```

If the typeinfo kind is `tkClass`, then the property information follows the `UnitName` string, as an array of `TPropInfo` records.

The `TPropData` record is not used, but is provided for completeness and compatibility with Delphi.

```
TPropData = packed record
  PropCount : Word;
  PropList : record end;
end;
```

The `TPropInfo` record describes one published property of a class:

```
PPropInfo = ^TPropInfo;
TPropInfo = packed record
  PropType : PTypeInfo;
  GetProc : Pointer;
  SetProc : Pointer;
  StoredProc : Pointer;
  Index : Integer;
  Default : Longint;
  NameIndex : SmallInt;
  PropProcs : Byte;
  Name : ShortString;
end;
```

The `Name` field is stored not with 255 characters, but with just as many characters as required to store the name.

```
TProcInfoProc = procedure(PropInfo : PPropInfo) of object;
```

The following pointer and array types are used for typecasts:

```
PPropList = ^TPropList;
TPropList = array[0..65535] of PPropInfo;
```

23.2 Function list by category

What follows is a listing of the available functions, grouped by category. For each function there is a reference to the page where the function can be found.

Examining published property information

Functions for retrieving or examining property information

Name	Description	Page
<code>FindPropInfo</code>	Getting property type information, With error checking.	477
<code>GetPropInfo</code>	Getting property type information, No error checking.	485
<code>GetPropInfos</code>	Find property information of a certain kind	??
<code>GetObjectPropClass</code>	Return the declared class of an object property	484
<code>GetPropList</code>	Get a list of all published properties	487

IsPublishedProp	Is a property published	490
IsStoredProp	Is a property stored	491
PropIsType	Is a property of a certain kind	492
PropType	Return the type of a property	493

Getting or setting property values

Functions to set or set a property's value.

Name	Description	Page
GetEnumProp	Return the value of an enumerated type property	478
GetFloatProp	Return the value of a float property	479
GetInt64Prop	Return the value of an Int64 property	480
GetMethodProp	Return the value of a procedural type property	481
GetObjectProp	Return the value of an object property	483
GetOrdProp	Return the value of an ordinal type property	484
GetProperty	Return the value of a property as a variant	488
GetSetProp	Return the value of a set property	488
GetStrProp	Return the value of a string property	489
GetVariantProp	Return the value of a variant property	490
SetEnumProp	Set the value of an enumerated type property	??
SetFloatProp	Set the value of a float property	??
SetInt64Prop	Set the value of an Int64 property	??
SetMethodProp	Set the value of a procedural type property	??
SetObjectProp	Set the value of an object property	??
SetOrdProp	Set the value of an ordinal type property	??
SetProperty	Set the value of a property through a variant	??
SetSetProp	Set the value of a set property	??
SetStrProp	Set the value of a string property	??
SetVariantProp	Set the value of a variant property	??

Auxiliary functions

Name	Description	Page
GetEnumName	Get an enumerated type element name	478
GetEnumValue	Get ordinal number of an enumerated type, based on the name.	479
GetTypeData	Skip type name and return a pointer to the type data	490
SetToString	Convert a set to its string representation	497
StringToSet	Convert a string representation of a set to a set	498

23.3 Functions and Procedures

FindPropInfo

Declaration: `Function FindPropInfo(AClass:TClass;const PropName: string): PPropInfo;`
`Function FindPropInfo(Instance: TObject; const PropName: string):`
`PPropInfo;`

Description: `FindPropInfo` examines the published property information of a class and returns a pointer to the property information for property `PropName`. The class to be examined can be specified in one of two ways:

AClass a class pointer.

Instance an instance of the class to be investigated.

If the property does not exist, a `EPropertyError` exception will be raised. The `GetPropInfo` (485) function has the same function as the `FindPropInfo` function, but returns `Nil` if the property does not exist.

Errors: Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetPropInfo` (485), `GetPropList` (487), `GetPropInfos` (486)

Listing: `typinfex/ex14.pp`

Program `example13;`

{ This program demonstrates the FindPropInfo function }

{ \$mode objfpc }

uses

`rttiobj, typinfo, sysutils;`

Var

`O : TMyTestObject;`

`PT : PTypeData;`

`PI : PPropInfo;`

`I, J : Longint;`

`PP : PPropList;`

`pri : PPropInfo;`

begin

`O:=TMyTestObject.Create;`

`PI:=FindPropInfo(O, 'BooleanField');`

`Writeln('FindPropInfo(Instance, BooleanField) : ', PI^.Name);`

`PI:=FindPropInfo(O.ClassType, 'ByteField');`

`Writeln('FindPropInfo(Class, ByteField) : ', PI^.Name);`

`Write('FindPropInfo(Class, NonExistingProp) : ');`

Try

`PI:=FindPropInfo(O, 'NonExistingProp');`

except

On `E: Exception do`

`Writeln('Caught exception "', E.ClassName, '" with message : ', E.Message);`

end;

`O.Free;`

end.

GetEnumName

Declaration: `Function GetEnumName(TypeInfo : PTypeInfo; Value : Integer) : string;`

Description: `GetEnumName` scans the type information for the enumeration type described by `TypeInfo` and returns the name of the enumeration constant for the element with ordinal value equal to `Value`.

If `Value` is out of range, the first element of the enumeration type is returned. The result is lower-cased, but this may change in the future.

This can be used in combination with `GetOrdProp` to stream a property of an enumerated type.

Errors: No check is done to determine whether `TypeInfo` really points to the type information for an enumerated type.

See also: `GetOrdProp` ([484](#)), `GetEnumValue` ([479](#))

Listing: `typinfex/ex9.pp`

```
program example9;

{ This program demonstrates the GetEnumName, GetEnumValue functions }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  TI : PTypeInfo;

begin
  O := TMyTestObject.Create;
  TI := GetPropInfo(O, 'MyEnumField')^.PropType;
  Writeln('GetEnumName      : ', GetEnumName(TI, Ord(O.MyEnumField)));
  Writeln('GetEnumValue(mefirst) : ', GetEnumName(TI, GetEnumValue(TI, 'mefirst')));
  O.Free;
end.
```

GetEnumProp

Declaration: `Function GetEnumProp(Instance: TObject; const PropInfo: PPropInfo): string;`
`Function GetEnumProp(Instance: TObject; const PropName: string): string;`

Description: `GetEnumProp` returns the value of an property of an enumerated type and returns the name of the enumerated value for the object `Instance`. The property whose value must be returned can be specified by its property info in `PropInfo` or by its name in `PropName`

Errors: No check is done to determine whether `PropInfo` really points to the property information for an enumerated type. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetEnumProp` ([494](#)), `GetOrdProp` ([484](#)), `GetStrProp` ([489](#)), `GetInt64Prop` ([480](#)), `GetMethodProp` ([481](#)), `GetSetProp` ([488](#)), `GetObjectProp` ([483](#)), `GetEnumProp` ([478](#))

Listing: `typinfex/ex2.pp`

```
program example2;

{ This program demonstrates the GetEnumProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  TI : PTypeInfo;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O, 'MyEnumField');
  TI:=PI^.PropType;
  Writeln('Enum property      : ');
  Writeln('Value                : ', GetEnumName(TI, Ord(O.MyEnumField)));
  Writeln('Get (name)                 : ', GetEnumProp(O, 'MyEnumField'));
  Writeln('Get (propinfo)             : ', GetEnumProp(O, PI));
  SetEnumProp(O, 'MyEnumField', 'meFirst');
  Writeln('Set (name, meFirst)        : ', GetEnumName(TI, Ord(O.MyEnumField)));
  SetEnumProp(O, PI, 'meSecond');
  Writeln('Set (propinfo, meSecond) : ', GetEnumName(TI, Ord(O.MyEnumField)));
  O.Free;
end.
```

GetEnumValue

Declaration: Function GetEnumValue(TypeInfo : PTypeInfo; const Name : string) : Integer;

Description: GetEnumValue scans the type information for the enumeration type described by TypeInfo and returns the ordinal value for the element in the enumerated type that has identifier Name. The identifier is searched in a case-insensitive manner.

This can be used to set the value of enumerated properties from a stream.

Errors: If Name is not found in the list of enumerated values, then -1 is returned. No check is done whether TypeInfo points to the type information for an enumerated type.

See also: GetEnumName ([478](#)), SetOrdProp ([495](#))

For an example, see GetEnumName ([478](#)).

GetFloatProp

Declaration: Function GetFloatProp(Instance : TObject; PropInfo : PPropInfo) : Extended;
Procedure SetFloatProp(Instance: TObject; const PropName: string;
Value: Extended);

Description: GetFloatProp returns the value of the float property described by PropInfo or with name Propname for the object Instance. All float types are converted to extended.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid float property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: SetFloatProp (494), GetOrdProp (484), GetStrProp (489), GetInt64Prop (480), GetMethodProp (481), GetSetProp (488), GetObjectProp (483), GetEnumProp (478)

Listing: typinfex/ex4.pp

```
program example4;

{ This program demonstrates the GetFloatProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Real property : ');
  PI:=GetPropInfo(O, 'RealField');
  Writeln('Value           : ', O.RealField);
  Writeln('Get (name)         : ', GetFloatProp(O, 'RealField'));
  Writeln('Get (propinfo)      : ', GetFloatProp(O, PI));
  SetFloatProp(O, 'RealField', system.Pi);
  Writeln('Set (name, pi)      : ', O.RealField);
  SetFloatProp(O, PI, exp(1));
  Writeln('Set (propinfo, e) : ', O.RealField);
  Writeln('Extended property : ');
  PI:=GetPropInfo(O, 'ExtendedField');
  Writeln('Value           : ', O.ExtendedField);
  Writeln('Get (name)         : ', GetFloatProp(O, 'ExtendedField'));
  Writeln('Get (propinfo)      : ', GetFloatProp(O, PI));
  SetFloatProp(O, 'ExtendedField', system.Pi);
  Writeln('Set (name, pi)      : ', O.ExtendedField);
  SetFloatProp(O, PI, exp(1));
  Writeln('Set (propinfo, e) : ', O.ExtendedField);
  O.Free;
end.
```

GetInt64Prop

Declaration: Function GetInt64Prop(Instance: TObject; PropInfo: PPropInfo): Int64;
Function GetInt64Prop(Instance: TObject; const PropName: string): Int64;

Description: *Publishing of Int64 properties is not yet supported by Free Pascal. This function is provided for Delphi compatibility only at the moment.*

GetInt64Prop returns the value of the property of type Int64 that is described by PropInfo or with name Propname for the object Instance.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid Int64 property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception

See also: SetInt64Prop (494), GetOrdProp (484), GetStrProp (489), GetFloatProp (479), GetMethodProp (481), GetSetProp (488), GetObjectProp (483), GetEnumProp (478)

Listing: typinfex/ex15.pp

```
program example15;

{ This program demonstrates the GetInt64Prop function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Int64 property : ');
  PI:=GetPropInfo(O, 'Int64Field');
  Writeln('Value           : ', O.Int64Field);
  Writeln('Get (name)         : ', GetInt64Prop(O, 'Int64Field'));
  Writeln('Get (propinfo)      : ', GetInt64Prop(O, PI));
  SetInt64Prop(O, 'Int64Field', 12345);
  Writeln('Set (name,12345)    : ', O.Int64Field);
  SetInt64Prop(O, PI, 54321);
  Writeln('Set (propinfo,54321) : ', O.Int64Field);
  O.Free;
end.
```

GetMethodProp

Declaration: Function GetMethodProp(Instance : TObject; PropInfo : PPropInfo) : TMethod;
Function GetMethodProp(Instance: TObject; const PropName: string): TMethod;

Description: GetMethodProp returns the method the property described by PropInfo or with name Propname for object Instance. The return type TMethod is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

Data points to the instance of the class with the method Code.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception

See also: SetMethodProp (495), GetOrdProp (484), GetStrProp (489), GetFloatProp (479), GetInt64Prop (480), GetSetProp (488), GetObjectProp (483), GetEnumProp (478)

Listing: typinfex/ex6.pp

```
program example6;

{ This program demonstrates the GetMethodProp function }

{$mode objfpc}

uses rttiobj, typinfo, sysutils;

Type
  TNotifyObject = Class(TObject)
    Procedure Notification1(Sender : TObject);
    Procedure Notification2(Sender : TObject);
  end;

Procedure TNotifyObject.Notification1(Sender : TObject);

begin
  Write('Received notification 1 of object with class: ');
  Writeln(Sender.ClassName);
end;

Procedure TNotifyObject.Notification2(Sender : TObject);

begin
  Write('Received notification 2 of object with class: ');
  Writeln(Sender.ClassName);
end;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO : TNotifyObject;
  M : TMethod;

Procedure PrintMethod (Const M : TMethod);

begin
  If (M.Data=Pointer(NO)) Then
    If (M.Code=Pointer(@TNotifyObject.Notification1)) then
      Writeln(' Notification1 ')
    else If (M.Code=Pointer(@TNotifyObject.Notification2)) then
      Writeln(' Notification2 ')
    else
      begin
        Write('Unknown method address (data: ');
        Write(hexStr(Longint(M.data),8));
        Writeln(' , code: ', hexstr(Longint(M.Code),8), ')');
      end;
    end;

end;

begin
  O:=TMyTestObject.Create;
  NO:=TNotifyObject.Create;
  O.NotifyEvent:=@NO.Notification1;
  PI:=GetPropInfo(O, 'NotifyEvent');
  Writeln('Method property : ');
  Write(' Notifying           : ');
```

```

O. Notify ;
Write ( ' Get ( name )                : ' );
M:=GetMethodProp(O, ' NotifyEvent ' );
PrintMethod (M);
Write ( ' Notifying                    : ' );
O. Notify ;
Write ( ' Get ( propinfo )            : ' );
M:=GetMethodProp(O, PI );
PrintMethod (M);
M.Data:=No;
M.Code:=Pointer(@NO. Notification2 );
SetMethodProp(O, ' NotifyEvent ',M);
Write ( ' Set ( name, Notification2 ) : ' );
M:=GetMethodProp(O, PI );
PrintMethod (M);
Write ( ' Notifying                    : ' );
O. Notify ;
Write ( ' Set ( propinfo, Notification1 ) : ' );
M.Data:=No;
M.Code:=Pointer(@NO. Notification1 );
SetMethodProp(O, PI ,M);
M:=GetMethodProp(O, PI );
PrintMethod (M);
Write ( ' Notifying                    : ' );
O. Notify ;
O. Free ;
end .

```

GetObjectProp

Declaration: Function GetObjectProp(Instance: TObject; const PropName: string): TObject;
Function GetObjectProp(Instance: TObject; const PropName: string; MinClass:TClass): TObject;
Function GetObjectProp(Instance: TObject; PropInfo: PPropInfo; MinClass:TClass): TObject;

Description: GetObjectProp returns the object which the property described by PropInfo with name Propname points to for object Instance.

If MinClass is specified, then if the object is not descendent of class MinClass, then Nil is returned.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: SetMethodProp (495), GetOrdProp (484), GetStrProp (489), GetFloatProp (479), GetInt64Prop (480), GetSetProp (488), GetObjectProp (483), GetEnumerator (478)

Listing: typinfex/ex5.pp

program example5;

{ This program demonstrates the GetObjectProp function }

```
{ $mode objfpc }

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO1, NO2 : TNamedObject;

begin
  O := TMyTestObject.Create;
  NO1 := TNamedObject.Create;
  NO1.ObjectName := 'First named object';
  NO2 := TNamedObject.Create;
  NO2.ObjectName := 'Second named object';
  O.ObjField := NO1;
  Writeln('Object property : ');
  PI := GetPropInfo(O, 'ObjField');
  Write('Property class      : ');
  Writeln(GetObjectPropClass(O, 'ObjField').ClassName);
  Write('Value                : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  Write('Get (name)              : ');
  Writeln((GetObjectProp(O, 'ObjField') as TNamedObject).ObjectName);
  Write('Get (propinfo)          : ');
  Writeln((GetObjectProp(O, PI, TObj) as TNamedObject).ObjectName);
  SetObjectProp(O, 'ObjField', NO2);
  Write('Set (name, NO2)         : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  SetObjectProp(O, PI, NO1);
  Write('Set (propinfo, NO1)    : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  O.Free;
end.
```

GetObjectPropClass

Declaration: Function GetObjectPropClass(Instance: TObject; const PropName: string): TClass;

Description: GetObjectPropClass returns the declared class of the property with name PropName. This may not be the actual class of the property value.

Errors: No checking is done whether Instance is non-nil. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: SetMethodProp ([495](#)), GetOrdProp ([484](#)), GetStrProp ([489](#)), GetFloatProp ([479](#)), GetInt64Prop ([480](#))

For an example, see GetObjectProp ([483](#)).

GetOrdProp

Declaration: Function GetOrdProp(Instance : TObject; PropInfo : PPropInfo) : Longint;
Function GetOrdProp(Instance: TObject; const PropName: string): Longint;

Description: `GetOrdProp` returns the value of the ordinal property described by `PropInfo` or with name `PropName` for the object `Instance`. The value is returned as a longint, which should be typecasted to the needed type.

Ordinal properties that can be retrieved include:

Integers and subranges of integersThe value of the integer will be returned.

Enumerated types and subranges of enumerated typesThe ordinal value of the enumerated type will be returned.

SetsIf the base type of the set has less than 31 possible values. If a bit is set in the return value, then the corresponding element of the base ordinal class of the set type must be included in the set.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetOrdProp` (495), `GetStrProp` (489), `GetFloatProp` (479), `GetInt64Prop` (480), `GetMethodProp` (481), `GetSetProp` (488), `GetObjectProp` (483), `GetEnumProp` (478)

Listing: `typinfex/ex1.pp`

```

program example1;

{ This program demonstrates the GetOrdProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln(' Boolean property      : ');
  Writeln(' Value                  : ',O.BooleanField);
  Writeln(' Ord(Value)              : ',Ord(O.BooleanField));
  Writeln(' Get (name)               : ',GetOrdProp(O, ' BooleanField' ));
  PI:=GetPropInfo(O, ' BooleanField' );
  Writeln(' Get (propinfo)           : ',GetOrdProp(O, PI));
  SetOrdProp(O, ' BooleanField', Ord( False ));
  Writeln(' Set (name, false)        : ',O.BooleanField);
  SetOrdProp(O, PI, Ord( True ));
  Writeln(' Set (propinfo, true)    : ',O.BooleanField);
  O.Free;
end.

```

GetPropInfo

Declaration: `Function GetPropInfo(AClass: TClass; const PropName: string; AKind: TTypeKinds) : PPropInfo;`
`Function GetPropInfo(AClass: TClass; const PropName: string): PPropInfo;`
`Function GetPropInfo(Instance: TObject; const PropName: string): PPropInfo;`
`Function GetPropInfo(Instance: TObject; const PropName: string; AKind: TTypeKinds) : PPropInfo;`

```
Function GetPropInfo(TypeInfo: PTypeInfo;const PropName: string) :  
PPropInfo;  
Function GetPropInfo(TypeInfo: PTypeInfo;const PropName: string; AKinds  
: TTypeKinds) : PPropInfo;
```

Description: GetPropInfo returns a pointer to the TPropInfo record for a the PropName property of a class. The class to examine can be specified in one of three ways:

InstanceAn instance of the class.

AClassA class pointer to the class.

TypeInfoA pointer to the type information of the class.

In each of these three ways, if AKinds is specified, if the property has TypeKind which is not included in AKinds, Nil will be returned.

Errors: If the property PropName does not exist, Nil is returned.

See also: GetPropInfos (486),GetPropList (487)

For an example, see most of the other functions.

GetPropInfos

Declaration: Procedure GetPropInfos(TypeInfo: PTypeInfo;PropList: PPropList);

Description: GetPropInfos stores pointers to the property information of all published properties of a class with class info TypeInfo in the list pointed to by PropList. The PropList pointer must point to a memory location that contains enough space to hold all properties of the class and its parent classes.

Errors: No checks are done to see whether PropList points to a memory area that is big enough to hold all pointers.

See also: GetPropInfo (485),GetPropList (487)

Listing: typinfex/ex12.pp

Program example12;

{ This program demonstrates the GetPropInfos function }

uses

rttiobj, typinfo;

Var

O : TMyTestObject;

PT : PTypeData;

PI : PTypeInfo;

I, J : Longint;

PP : PPropList;

pri : PPropInfo;

begin

O:=TMyTestObject.Create;

PI:=O.ClassInfo;

PT:=GetTypeData(PI);


```
WriteLn('Property Count : ',PT^.PropCount);
GetMem (PP,PT^.PropCount*SizeOf(Pointer));
GetPropInfos(PI,PP);
For I:=0 to PT^.PropCount-1 do
begin
  With PP^[i]^ do
  begin
    Write('Property ',i+1:3,' : ',name:30);
    writeLn('  Type : ',TypeNames[typinfo.PropType(O,Name)]);
  end;
end;
FreeMem(PP);
O.Free;
end.
```

GetPropList

Declaration: Function GetPropList(TypeInfo : PTypeInfo; TypeKinds : TTypeKinds; PropList : PPropList) : Integer;

Description: GetPropList stores pointers to property information of the class with class info TypeInfo for properties of kind TypeKinds in the list pointed to by PropList. PropList must contain enough space to hold all properties.

The function returns the number of pointers that matched the criteria and were stored in PropList.

Errors: No checks are done to see whether PropList points to a memory area that is big enough to hold all pointers.

See also: GetPropInfos ([486](#)), GetPropInfo ([485](#))

Listing: typinfex/ex13.pp

Program example13;

{ This program demonstrates the GetPropList function }

uses

rttiobj, typinfo;

Var

O : TMyTestObject;

PT : PTypeData;

PI : PTypeInfo;

I, J : Longint;

PP : PPropList;

pri : PPropInfo;

begin

O:=TMyTestObject.Create;

PI:=O.ClassInfo;

PT:=GetTypeData(PI);

WriteLn('Total property Count : ',PT^.PropCount);

GetMem (PP,PT^.PropCount*SizeOf(Pointer));

J:=GetPropList(PI, OrdinalTypes, PP);

WriteLn('Ordinal property Count : ',J);

For I:=0 to J-1 do

```
begin
  With PP^[i]^ do
    begin
      Write(' Property ', i+1:3, ': ', name:30);
      writeln('   Type: ', TypeName[typinfo.PropType(O,Name)]);
    end;
  end;
  FreeMem(PP);
  O.Free;
end.
```

GetPropValue

Declaration: Function GetPropValue(Instance: TObject; const PropName: string): Variant;
Function GetPropValue(Instance: TObject; const PropName: string; PreferStrings: Boolean): Variant;

Description: Due to missing Variant support, GetPropValue is not yet implemented. The declaration is provided for compatibility with Delphi.

Errors:

See also:

GetSetProp

Declaration: Function GetSetProp(Instance: TObject; const PropInfo: PPropInfo; Brackets: Boolean): string;
Function GetSetProp(Instance: TObject; const PropName: string): string;
Function GetSetProp(Instance: TObject; const PropName: string; Brackets: Boolean): string;

Description: GetSetProp returns the contents of a set property as a string. The property to be returned can be specified by its name in PropName or by its property information in PropInfo.

The returned set is a string representation of the elements in the set as returned by [SetToString \(497\)](#). The Brackets option can be used to enclose the string representation in square brackets.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid ordinal property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: SetSetProp ([496](#)), GetStrProp ([489](#)), GetFloatProp ([479](#)), GetInt64Prop ([480](#)), GetMethodProp ([481](#))

Listing: typinfex/ex7.pp

```
program example7;

{ This program demonstrates the GetSetProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
```

```

O : TMyTestObject;
PI : PPropInfo;

Function SetAsString (ASet : TMyEnums) : String;

Var
  i : TmyEnum;

begin
  result:= '';
  For i:=mefirst to methird do
    If i in ASet then
      begin
        If (Result<>'') then
          Result:=Result+', ' ;
        Result:=Result+MyEnumNames[ i ];
      end;
  end;

Var
  S : TMyEnums;

begin
  O:=TMyTestObject.Create;
  O.SetField :=[ mefirst,meSecond,meThird ];
  Writeln('Set property      : ');
  Writeln('Value                               : ',SetAsString(O.SetField));
  Writeln('Ord(Value)                             : ',Longint(O.SetField));
  Writeln('Get (name)                               : ',GetSetProp(O,'SetField'));
  PI:=GetPropInfo(O,'SetField');
  Writeln('Get (propinfo)                           : ',GetSetProp(O,PI,false));
  S:=[ meFirst,meThird ];
  SetOrdProp(O,'SetField',Integer(S));
  Write('Set (name,[ mefirst, methird ]) : ');
  Writeln(SetAsString(O.SetField));
  S:=[ meSecond ];
  SetOrdProp(O,PI,Integer(S));
  Write('Set (propinfo,[ meSecond ]) : ');
  Writeln(SetAsString(O.SetField));
  O.Free;
end.

```

GetStrProp

Declaration: `Function GetStrProp(Instance : TObject; PropInfo : PPropInfo) : Ansistring;`
`Function GetStrProp(Instance: TObject; const PropName: string): string;`

Description: `GetStrProp` returns the value of the string property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetStrProp` (497), `GetOrdProp` (484), `GetFloatProp` (479), `GetInt64Prop` (480), `GetMethodProp` (481)

Listing: `typinfex/ex3.pp`

```
program example3;

{ This program demonstrates the GetStrProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O, 'AnsiStringField');
  Writeln('String property : ');
  Writeln('Value           : ', O.AnsiStringField);
  Writeln('Get (name)         : ', GetStrProp(O, 'AnsiStringField'));
  Writeln('Get (propinfo)      : ', GetStrProp(O, PI));
  SetStrProp(O, 'AnsiStringField', 'First');
  Writeln('Set (name, ''First'') : ', O.AnsiStringField);
  SetStrProp(O, PI, 'Second');
  Writeln('Set (propinfo, ''Second'') : ', O.AnsiStringField);
  O.Free;
end.
```

GetTypeData

Declaration: `Function GetTypeData(TypeInfo : PTypeInfo) : PTypeData;`

Description: `GetTypeData` returns a pointer to the `TTypeData` record that follows after the `TTypeInfo` record pointed to by `TypeInfo`. It essentially skips the `Kind` and `Name` fields in the `TTypeInfo` record.

Errors: None.

See also:

GetVariantProp

Declaration: `Function GetVariantProp(Instance : TObject; PropInfo : PPropInfo): Variant;`

Description: Due to missing Variant support, the `GetVariantProp` function is not yet implemented. Provided for Delphi compatibility only.

Errors:

See also: `SetVariantProp` ([498](#))

IsPublishedProp

Declaration: `Function IsPublishedProp(AClass: TClass; const PropName: string): Boolean;`
`Function IsPublishedProp(Instance: TObject; const PropName: string): Boolean;`

Description: `IsPublishedProp` returns true if a class has a published property with name `PropName`. The class can be specified in one of two ways:

AClassA class pointer to the class.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsStoredProp` ([491](#)), `PropsType` ([492](#))

Listing: `typinfex/ex10.pp`

```
program example10;

{ This program demonstrates the IsPublishedProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests      : ');
  Write('IsPublishedProp(O, BooleanField)      : ');
  Writeln(IsPublishedProp(O, 'BooleanField'));
  Write('IsPublishedProp(Class, BooleanField) : ');
  Writeln(IsPublishedProp(O.ClassType, 'BooleanField'));
  Write('IsPublishedProp(O, SomeField)           : ');
  Writeln(IsPublishedProp(O, 'SomeField'));
  Write('IsPublishedProp(Class, SomeField)       : ');
  Writeln(IsPublishedProp(O.ClassType, 'SomeField'));
  O.Free;
end.
```

IsStoredProp

Declaration: `Function IsStoredProp(Instance : TObject; PropInfo : PPropInfo) : Boolean;`
`Function IsStoredProp(Instance: TObject; const PropName: string): Boolean;`

Description: `IsStoredProp` returns True if the `Stored` modifier evaluates to True for the property described by `PropInfo` or with name `PropName` for object `Instance`. It returns False otherwise. If the function returns True, this indicates that the property should be written when streaming the object `Instance`.

If there was no stored modifier in the declaration of the property, True will be returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` ([490](#)), `PropsType` ([492](#))

Listing: typinfex/ex11.pp

```
program example11;

{ This program demonstrates the IsStoredProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Stored tests      : ');
  Write('IsStoredProp(O, StoredIntegerConstFalse)      : ');
  Writeln(IsStoredProp(O, 'StoredIntegerConstFalse'));
  Write('IsStoredProp(O, StoredIntegerConstTrue)      : ');
  Writeln(IsStoredProp(O, 'StoredIntegerConstTrue'));
  Write('IsStoredProp(O, StoredIntegerMethod)      : ');
  Writeln(IsStoredProp(O, 'StoredIntegerMethod'));
  Write('IsStoredProp(O, StoredIntegerVirtualMethod) : ');
  Writeln(IsStoredProp(O, 'StoredIntegerVirtualMethod'));
  O.Free;
end.
```

PropIsType

Declaration: Function PropIsType(AClass: TClass; const PropName: string; TypeKind: TTypeKind): Boolean;
Function PropIsType(Instance: TObject; const PropName: string; TypeKind: TTypeKind): Boolean;

Description: PropIsType returns True if the property with name PropName has type TypeKind. It returns False otherwise. The class to be examined can be specified in one of two ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure Instance or AClass are valid pointers. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: IsPublishedProp ([490](#)), IsStoredProp ([491](#)), PropType ([493](#))

Listing: typinfex/ex16.pp

```
program example16;

{ This program demonstrates the PropIsType function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
```

```
begin
  O:=TMyTestObject.Create;
  Writeln('Property tests      : ');
  Write(' PropIsType(O, BooleanField, tkBool)      : ');
  Writeln(PropIsType(O, ' BooleanField ', tkBool));
  Write(' PropIsType(Class, BooleanField, tkBool) : ');
  Writeln(PropIsType(O.ClassType, ' BooleanField ', tkBool));
  Write(' PropIsType(O, ByteField, tkString)      : ');
  Writeln(PropIsType(O, ' ByteField ', tkString));
  Write(' PropIsType(Class, ByteField, tkString) : ');
  Writeln(PropIsType(O.ClassType, ' ByteField ', tkString));
  O.Free;
end.
```

PropType

Declaration: `Function PropType(AClass: TClass; const PropName: string): TTypeKind;`
`Function PropType(Instance: TObject; const PropName: string): TTypeKind;`

Description: `PropType` returns the type of the property `PropName` for a class. The class to be examined can be specified in one of 2 ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` ([490](#)), `IsStoredProp` ([491](#)), `PropIsType` ([492](#))

Listing: `typinfex/ex17.pp`

```
program example17;

{ This program demonstrates the PropType function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests      : ');
  Write(' PropType(O, BooleanField)      : ');
  Writeln(TypeNames[PropType(O, ' BooleanField ')]);
  Write(' PropType(Class, BooleanField) : ');
  Writeln(TypeNames[PropType(O.ClassType, ' BooleanField ')]);
  Write(' PropType(O, ByteField)      : ');
  Writeln(TypeNames[PropType(O, ' ByteField ')]);
  Write(' PropType(Class, ByteField) : ');
  Writeln(TypeNames[PropType(O.ClassType, ' ByteField ')]);
  O.Free;
end.
```

SetEnumProp

Declaration: `Procedure SetEnumProp(Instance: TObject; const PropInfo: PPropInfo; const Value: string);`
`Procedure SetEnumProp(Instance: TObject; const PropName: string; const Value: string);`

Description: `SetEnumProp` sets the property described by `PropInfo` or with name `PropName` to `Value`. `Value` must be a string with the name of the enumerate value, i.e. it can be used as an argument to `GetEnumValue` (479).

Errors: No checks are done to ensure `Instance` or `PropInfo` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetEnumProp` (478), `SetStrProp` (497), `SetFloatProp` (494), `SetInt64Prop` (494), `SetMethodProp` (495).

For an example, see `GetEnumProp` (478).

SetFloatProp

Declaration: `Procedure SetFloatProp(Instance : TObject; PropInfo : PPropInfo; Value : Extended);`
`Procedure SetFloatProp(Instance: TObject; const PropName: string; Value: Extended);`

Description: `SetFloatProp` assigns `Value` to the property described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetFloatProp` (479), `SetOrdProp` (495), `SetStrProp` (497), `SetInt64Prop` (494), `SetMethodProp` (495)

For an example, see `GetFloatProp` (479).

SetInt64Prop

Declaration: `Procedure SetInt64Prop(Instance: TObject; PropInfo: PPropInfo; const Value: Int64);`
`Procedure SetInt64Prop(Instance: TObject; const PropName: string; const Value: Int64);`

Description: `SetInt64Prop` assigns `Value` to the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetInt64Prop` (480), `GetMethodProp` (481), `SetOrdProp` (495), `SetStrProp` (497), `SetFloatProp` (494)

For an example, see `GetInt64Prop` (480).

SetMethodProp

Declaration: `Procedure SetMethodProp(Instance : TObject; PropInfo : PPropInfo; const Value : TMethod);`
`Procedure SetMethodProp(Instance: TObject; const PropName: string; const Value: TMethod);`

Description: SetMethodProp assigns Value to the method the property described by PropInfo or with name Propname for object Instance.

The type TMethod of the Value parameter is defined in the SysUtils unit as:

```
TMethod = packed record
    Code, Data: Pointer;
end;
```

Data should point to the instance of the class with the method Code.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: [GetMethodProp \(481\)](#), [SetOrdProp \(495\)](#), [SetStrProp \(497\)](#), [SetFloatProp \(494\)](#), [SetInt64Prop \(494\)](#)

For an example, see [GetMethodProp \(481\)](#).

SetObjectProp

Declaration: `Procedure SetObjectProp(Instance: TObject; PropInfo: PPropInfo; Value: TObject);`
`Procedure SetObjectProp(Instance: TObject; const PropName: string; Value: TObject);`

Description: SetObjectProp assigns Value to the the object property described by PropInfo or with name Propname for the object Instance.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: [GetObjectProp \(483\)](#), [SetOrdProp \(495\)](#), [SetStrProp \(497\)](#), [SetFloatProp \(494\)](#), [SetInt64Prop \(494\)](#), [SetMethodProp \(495\)](#)

For an example, see [GetObjectProp \(483\)](#).

SetOrdProp

Declaration: `Procedure SetOrdProp(Instance : TObject; PropInfo : PPropInfo; Value : Longint);`
`Procedure SetOrdProp(Instance: TObject; const PropName: string; Value: Longint);`

Description: SetOrdProp assigns Value to the the ordinal property described by PropInfo or with name Propname for the object Instance.

Ordinal properties that can be set include:

Integers and subranges of integersThe actual value of the integer must be passed.

Enumerated types and subranges of enumerated typesThe ordinal value of the enumerated type must be passed.

Subrange typesof integers or enumerated types. Here the ordinal value must be passed.

SetsIf the base type of the set has less than 31 possible values. For each possible value; the corresponding bit of Value must be set.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid ordinal property of Instance. No range checking is performed. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetOrdProp (484), SetStrProp (497), SetFloatProp (494), SetInt64Prop (494), SetMethodProp (495)

For an example, see GetOrdProp (484).

SetPropValue

Declaration: Procedure SetPropValue(Instance: TObject; const PropName: string; const Value: Variant);

Description: Due to missing Variant support, this function is not yet implemented; it is provided for Delphi compatibility only.

Errors:

See also:

SetSetProp

Declaration: Procedure SetSetProp(Instance: TObject; const PropInfo: PPropInfo; const Value: string);
Procedure SetSetProp(Instance: TObject; const PropName: string; const Value: string);

Description: SetSetProp sets the property specified by PropInfo or PropName for object Instance to Value. Value is a string which contains a comma-separated list of values, each value being a string-representation of the enumerated value that should be included in the set. The value should be accepted by the StringToSet (498) function.

The value can be formed using the SetToString (497) function.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid ordinal property of Instance. No range checking is performed. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetSetProp (488), SetOrdProp (495), SetStrProp (497), SetFloatProp (494), SetInt64Prop (494), SetMethodProp (495), SetToString (497), StringToSet (498)

For an example, see GetSetProp (488).

SetStrProp

Declaration: `procedure SetStrProp(Instance : TObject; PropInfo : PPropInfo; const Value : Ansistring);`
`Procedure SetStrProp(Instance: TObject; const PropName: string; const Value: AnsiString);`

Description: SetStrProp assigns Value to the string property described by PropInfo or with name Propname for object Instance.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid string property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetStrProp ([489](#)), SetOrdProp ([495](#)), SetFloatProp ([494](#)), SetInt64Prop ([494](#)), SetMethodProp ([495](#))

For an example, see GetStrProp ([489](#))

SetToString

Declaration: `function SetToString(PropInfo: PPropInfo; Value: Integer) : String;`
`function SetToString(PropInfo: PPropInfo; Value: Integer; Brackets: Boolean) : String;`

Description: SetToString takes an integer representation of a set (as received e.g. by GetOrdProp) and turns it into a string representing the elements in the set, based on the type information found in the PropInfo property information. By default, the string representation is not surrounded by square brackets. Setting the Brackets parameter to True will surround the string representation with brackets.

The function returns the string representation of the set.

Errors: No checking is done to see whether PropInfo points to valid property information.

See also: GetEnumName ([478](#)), GetEnumValue ([479](#)), StringToSet ([498](#))

Listing: typinfex/ex18.pp

```
program example18;

{ This program demonstrates the SetToString function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  I : longint;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O, 'SetField');
  O.SetField:=[ mefirst, meSecond, meThird ];
  I:=GetOrdProp(O, PI);
  Writeln('Set property to string : ');
```

```
WriteLn('Value  : ', SetToString(PI, I, False));
O.SetField := [ mefirst, meSecond];
I := GetOrdProp(O, PI);
WriteLn('Value  : ', SetToString(PI, I, True));
I := StringToSet(PI, 'mefirst');
SetOrdProp(O, PI, I);
I := GetOrdProp(O, PI);
WriteLn('Value  : ', SetToString(PI, I, False));
I := StringToSet(PI, '[ mesecond, methird ]');
SetOrdProp(O, PI, I);
I := GetOrdProp(O, PI);
WriteLn('Value  : ', SetToString(PI, I, True));
O.Free;
end.
```

SetVariantProp

Declaration: `Procedure SetVariantProp(Instance : TObject; PropInfo : PPropInfo; Const Value: Variant);`
`Procedure SetVariantProp(Instance: TObject; const PropName: string; const Value: Variant);`

Description: Due to missing Variant support, this function is not yet implemented. Provided for Delphi compatibility only.

Errors:

See also:

StringToSet

Declaration: `function StringToSet(PropInfo: PPropInfo; const Value: string): Integer;`

Description: `StringToSet` converts the string representation of a set in `Value` to a integer representation of the set, using the property information found in `PropInfo`. This property information should point to the property information of a set property. The function returns the integer representation of the set. (i.e, the set value, typecast to an integer)

The string representation can be surrounded with square brackets, and must consist of the names of the elements of the base type of the set. The base type of the set should be an enumerated type. The elements should be separated by commas, and may be surrounded by spaces. each of the names will be fed to the `GetEnumValue` ([479](#)) function.

Errors: No checking is done to see whether `PropInfo` points to valid property information. If a wrong name is given for an enumerated value, then an `EPropertyError` will be raised.

See also: `GetEnumName` ([478](#)), `GetEnumValue` ([479](#)), `SetToString` ([497](#))

For an example, see `SetToString` ([497](#)).

Chapter 24

The VIDEO unit

The Video unit implements a screen access layer which is system independent. It can be used to write on the screen in a system-independent way, which should be optimal on all platforms for which the unit is implemented.

The working of the Video is simple: After calling `InitVideo` (509), the array `VideoBuf` contains a representation of the video screen of size `ScreenWidth*ScreenHeight`, going from left to right and top to bottom when walking the array elements: `VideoBuf[0]` contains the character and color code of the top-left character on the screen. `VideoBuf[ScreenWidth]` contains the data for the character in the first column of the second row on the screen, and so on.

To write to the 'screen', the text to be written should be written to the `VideoBuf` array. Calling `UpdateScreen` (512) will then cp the text to the screen in the most optimal way. (an example can be found further on).

The color attribute is a combination of the foreground and background color, plus the blink bit. The bits describe the various color combinations:

bits 0-3 The foreground color. Can be set using all color constants.

bits 4-6 The background color. Can be set using a subset of the color constants.

bit 7 The blinking bit. If this bit is set, the character will appear blinking.

Each possible color has a constant associated with it, see page 500 for a list of constants.

The contents of the `VideoBuf` array may be modified: This is 'writing' to the screen. As soon as everything that needs to be written in the array is in the `VideoBuf` array, calling `UpdateScreen` will copy the contents of the array screen to the screen, in a manner that is as efficient as possible.

The updating of the screen can be prohibited to optimize performance; To this end, the `LockScreenUpdate` (509) function can be used: This will increment an internal counter. As long as the counter differs from zero, calling `UpdateScreen` (512) will not do anything. The counter can be lowered with `UnlockScreenUpdate` (512). When it reaches zero, the next call to `UpdateScreen` (512) will actually update the screen. This is useful when having nested procedures that do a lot of screen writing.

The video unit also presents an interface for custom screen drivers, thus it is possible to override the default screen driver with a custom screen driver, see the `SetVideoDriver` (511) call. The current video driver can be retrieved using the `GetVideoDriver` (507) call.

Remark: The video unit should *not* be used together with the crt unit. Doing so will result in very strange behaviour, possibly program crashes.

24.1 Constants, Type and variables

Constants

The following constants describe colors that can be used as foreground and background colors.

```
Black      = 0;
Blue       = 1;
Green      = 2;
Cyan       = 3;
Red        = 4;
Magenta    = 5;
Brown      = 6;
LightGray  = 7;
```

The following color constants can be used as foreground colors only:

```
DarkGray   = 8;
LightBlue  = 9;
LightGreen = 10;
LightCyan  = 11;
LightRed   = 12;
LightMagenta = 13;
Yellow     = 14;
White      = 15;
```

The foreground and background color can be combined to a color attribute with the following code:

```
Attr:=ForeGroundColor + (BackGroundColor shl 4);
```

The color attribute can be logically or-ed with the blink attribute to produce a blinking character:

```
Blink      = 128;
```

But not all drivers may support this.

The following constants describe the capabilities of a certain video mode:

```
cpUnderLine = $0001;
cpBlink     = $0002;
cpColor     = $0004;
cpChangeFont = $0008;
cpChangeMode = $0010;
cpChangeCursor = $0020;
```

The following constants describe the various supported cursor modes:

```
crHidden     = 0;
crUnderLine  = 1;
crBlock      = 2;
crHalfBlock  = 3;
```

When a video function needs to report an error condition, the following constants are used:

```
vioOK          = 0;
errVioBase     = 1000;
errVioInit     = errVioBase + 1; { Initialization error}
errVioNotSupported = errVioBase + 2; { Unsupported function }
errVioNoSuchMode = errVioBase + 3; { No such video mode }
```

The following constants can be read to get some information about the current screen:

```
ScreenWidth    : Word = 0;
ScreenHeight   : Word = 0;
LowAscii       : Boolean = true;
NoExtendedFrame : Boolean = false;
FVMaxWidth     = 132;
```

The error-handling code uses the following constants:

```
errOk          = 0;
ErrorCode: Longint = ErrOK;
ErrorInfo: Pointer = nil;
ErrorHandler: TErrorHandler = DefaultErrorHandler;
```

The ErrorHandler variable can be set to a custom-error handling function. It is set by default to the `DefaultErrorHandler` ([504](#)) function.

Types

The TVideoMode record describes a videomode. Its fields are self-explaining: Col , Row describe the number of columns and rows on the screen for this mode. Color is True if this mode supports colors, or False if not.

```
PVideoMode = ^TVideoMode;
TVideoMode = record
    Col,Row : Word;
    Color   : Boolean;
end;
```

TVideoCell describes one character on the screen. The high byte contains the color attribute with which the character is drawn on the screen, and the low byte contains the ASCII code of the character to be drawn.

```
TVideoCell = Word;
PVideoCell = ^TVideoCell;
```

The TVideoBuf and PVideoBuf are two types used to represent the screen.

```
TVideoBuf = array[0..32759] of TVideoCell;
PVideoBuf = ^TVideoBuf;
```

The following type is used when reporting error conditions:

```
TErrorHandlerReturnValue = (errRetry, errAbort, errContinue);
```

Here, errRetry means retry the operation, errAbort abort and return error code and errContinue means abort without returning an errorcode.

The TErrorHandler function is used to register an own error handling function. It should be used when installing a custom error handling function, and must return one of the above values.

```
TErrorHandler =  
  function (Code: Longint; Info: Pointer): TErrorHandlerReturnValue;
```

Code should contain the error code for the error condition, and the Info parameter may contain any data type specific to the error code passed to the function.

The TVideoDriver record can be used to install a custom video driver, with the SetVideoDriver (511) call:

```
TVideoDriver = Record  
  InitDriver      : Procedure;  
  DoneDriver      : Procedure;  
  UpdateScreen    : Procedure(Force : Boolean);  
  ClearScreen     : Procedure;  
  SetVideoMode    : Function (Const Mode : TVideoMode) : Boolean;  
  GetVideoModeCount : Function : Word;  
  GetVideoModeData : Function(Index : Word; Var Data : TVideoMode) : Boolean;  
  SetCursorPos    : procedure (NewCursorX, NewCursorY: Word);  
  GetCursorType   : function : Word;  
  SetCursorType   : procedure (NewType: Word);  
  GetCapabilities : Function : Word;  
end;
```

Variables

The following variables contain information about the current screen status:

```
ScreenColor      : Boolean;  
CursorX, CursorY : Word;
```

ScreenColor indicates whether the current screen supports colors. CursorX, CursorY contain the current cursor position.

The following variables form the heart of the Video unit: The VideoBuf array represents the physical screen. Writing to this array and calling UpdateScreen (512) will write the actual characters to the screen. VideoBufSize contains the actual screen size, and is equal to the product of the number of columns times the number of lines on the screen (ScreenWidth*ScreenHeight).

```
VideoBuf      : PVideoBuf;  
OldVideoBuf   : PVideoBuf;  
VideoBufSize  : Longint;
```

The OldVideoBuf contains the state of the video screen after the last screen update. The UpdateScreen (512) function uses this array to decide which characters on screen should be updated, and which not.

Note that the OldVideoBuf array may be ignored by some drivers, so it should not be used. The Array is in the interface section of the video unit mainly so drivers that need it can make use of it.

24.2 Functions and Procedures

The examples in this section make use of the unit vidutil, which contains the TextOut function. This function writes a text to the screen at a given location. It looks as follows:

Listing: videoex/vidutil.pp

```
unit vidutil;  
  
Interface  
  
uses  
    video;  
  
Procedure TextOut(X,Y : Word;Const S : String);  
  
Implementation  
  
Procedure TextOut(X,Y : Word;Const S : String);  
  
Var  
    W,P,I,M : Word;  
  
begin  
    P:=((X-1)+(Y-1)*ScreenWidth);  
    M:=Length(S);  
    If P+M>ScreenWidth*ScreenHeight then  
        M:=ScreenWidth*ScreenHeight-P;  
    For I:=1 to M do  
        VideoBuf^[P+I-1]:=Ord(S[I])+($07 shr 8);  
end;  
  
end.
```

ClearScreen

Declaration: `procedure ClearScreen;`

Description: `ClearScreen` clears the entire screen, and calls `UpdateScreen` (512) after that. This is done by writing spaces to all character cells of the video buffer in the default color (lightgray on black, color attribute \$07).

Errors: None.

See also: `InitVideo` (509), `UpdateScreen` (512)

Listing: `videoex/ex3.pp`

```
program testvideo;  
  
uses video, keyboard, vidutil;  
  
Var  
    i : longint;  
    k : TKeyEvent;  
  
begin  
    InitVideo;  
    InitKeyboard;  
    For I:=1 to 10 do  
        TextOut(i,i, 'Press any key to clear screen');  
        UpdateScreen(false);  
        K:=GetKeyEvent;  
        ClearScreen;
```

```
TextOut(1,1,'Cleared screen. Press any key to end');
UpdateScreen(true);
K:=GetKeyEvent;
DoneKeyBoard;
DoneVideo;
end.
```

DefaultErrorHandler

Declaration: `function DefaultErrorHandler(AErrorCode: Longint; AErrorInfo: Pointer): TErrorHandlerReturnValue;`

Description: `DefaultErrorHandler` is the default error handler used by the video driver. It simply sets the error code `AErrorCode` and `AErrorInfo` in the global variables `ErrorCode` and `ErrorInfo` and returns `errContinue`.

Errors: None.

See also:

DoneVideo

Declaration: `procedure DoneVideo;`

Description: `DoneVideo` disables the Video driver if the video driver is active. If the videodriver was already disabled or not yet initialized, it does nothing. Disabling the driver means it will clean up any allocated resources, possibly restore the screen in the state it was before `InitVideo` was called. Particularly, the `VideoBuf` and `OldVideoBuf` arrays are no longer valid after a call to `DoneVideo`.

The `DoneVideo` should always be called if `InitVideo` was called. Failing to do so may leave the screen in an unusable state after the program exits.

Errors: Normally none. If the driver reports an error, this is done through the `ErrorCode` variable.

See also: `InitVideo` ([509](#))

For an example, see most other functions.

GetCapabilities

Declaration: `function GetCapabilities: Word;`

Description: `GetCapabilities` returns the capabilities of the current driver. It is an or-ed combination of the following constants:

cpUnderLineThe driver supports underlined characters.

cpBlinkThe driver supports blinking characters.

cpColorThe driver supports colors.

cpChangeFontThe driver supports the setting of a screen font. Note, however, that a font setting API is not supported by the video unit.

cpChangeModeThe driver supports the setting of screen modes.

cpChangeCursorThe driver supports changing the cursor shape.

Note that the video driver should not yet be initialized to use this function. It is a property of the driver.

Errors: None.

See also: [GetCursorType \(505\)](#), [GetVideoDriver \(507\)](#)

Listing: videoex/ex4.pp

Program Example4;

{ Program to demonstrate the GetCapabilities function. }

Uses video;

Var

W: Word;

Procedure TestCap(Cap: Word; Msg : **String**);

begin

Write(Msg, ' : ');

If (W **and** Cap=Cap) **then**

Writeln('Yes')

else

Writeln('No');

end;

begin

W:=GetCapabilities;

Writeln('Video driver supports following functionality');

 TestCap(cpUnderLine, 'Underlined characters');

 TestCap(cpBlink, 'Blinking characters');

 TestCap(cpColor, 'Color characters');

 TestCap(cpChangeFont, 'Changing font');

 TestCap(cpChangeMode, 'Changing video mode');

 TestCap(cpChangeCursor, 'Changing cursor shape');

end.

GetCursorType

Declaration: function GetCursorType: Word;

Description: `GetCursorType` returns the current cursor type. It is one of the following values:

crHiddenThe cursor is currently hidden.

crUnderLineThe cursor is currently the underline character.

crBlockThe cursor is currently the block character.

crHalfBlockThe cursor is currently a block with height of half the character cell height.

Note that not all drivers support all types of cursors.

Errors: None.

See also: [SetCursorType \(511\)](#), [GetCapabilities \(504\)](#)

Listing: videoex/ex5.pp

```
Program Example5;

{ Program to demonstrate the GetCursorType function. }

Uses video, keyboard, vidutil;

Const
  Cursortypes : Array[crHidden..crHalfBlock] of string =
    ('Hidden', 'UnderLine', 'Block', 'HalfBlock');

begin
  InitVideo;
  InitKeyboard;
  TextOut(1,1, 'Cursor type: '+CursorTypes[GetCursorType]);
  TextOut(1,2, 'Press any key to exit. ');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

GetLockScreenCount

Declaration: `Function GetLockScreenCount : integer;`

Description: `GetLockScreenCount` returns the current lock level. When the lock level is zero, a call to `UpdateScreen` ([512](#)) will actually update the screen.

Errors: None.

See also: `LockScreenUpdate` ([509](#)), `UnlockScreenUpdate` ([512](#)), `UpdateScreen` ([512](#))

Listing: videoex/ex6.pp

```
Program Example6;

{ Program to demonstrate the GetLockScreenCount function. }

Uses video, keyboard, vidutil;

Var
  I : Longint;
  S : String;

begin
  InitVideo;
  InitKeyboard;
  TextOut(1,1, 'Press key till new text appears. ');
  UpdateScreen(False);
  Randomize;
  For I:=0 to Random(10)+1 do
    LockScreenUpdate;
  I:=0;
  While GetLockScreenCount<>0 do
    begin
      Inc(I);
      Str(I, S);
```

```
UnlockScreenUpdate;
GetKeyEvent;
TextOut(1,1,'UnLockScreenUpdate had to be called '+S+' times');
UpdateScreen(False);
end;
TextOut(1,2,'Press any key to end. ');
UpdateScreen(False);
GetKeyEvent;
DoneKeyboard;
DoneVideo;
end.
```

GetVideoDriver

Declaration: `Procedure GetVideoDriver (Var Driver : TVideoDriver);`

Description: `GetVideoDriver` retrieves the current videodriver and returns it in `Driver`. This can be used to chain video drivers.

Errors: None.

See also: `SetVideoDriver` ([511](#))

For an example, see the section on writing a custom video driver.

GetVideoMode

Declaration: `procedure GetVideoMode(var Mode: TVideoMode);`

Description: `GetVideoMode` returns the settings of the currently active video mode. The `row`, `col` fields indicate the dimensions of the current video mode, and `Color` is true if the current video supports colors.

Errors: None.

See also: `SetVideoMode` ([511](#)), `GetVideoModeData` ([509](#))

Listing: videoex/ex7.pp

Program Example7;

{ Program to demonstrate the GetVideoMode function. }

Uses video, keyboard, vidutil;

Var

 M : TVideoMode;

 S : **String**;

begin

 InitVideo;

 InitKeyboard;

 GetVideoMode(M);

if M.Color **then**

 TextOut(1,1,'Current mode has color')

else

 TextOut(1,1,'Current mode does not have color');

```
Str(M.Row,S);
TextOut(1,2,'Number of rows    : '+S);
Str(M.Col,S);
TextOut(1,3,'Number of columns : '+S);
Textout(1,4,'Press any key to exit. ');
UpdateScreen(False);
GetKeyEvent;
DoneKeyboard;
DoneVideo;
end.
```

GetVideoModeCount

Declaration: Function GetVideoModeCount : Word;

Description: GetVideoModeCount returns the number of video modes that the current driver supports. If the driver does not support switching of modes, then 1 is returned.

This function can be used in conjunction with the GetVideoModeData ([509](#)) function to retrieve data for the supported video modes.

Errors: None.

See also: GetVideoModeData ([509](#)), GetVideoMode ([507](#))

Listing: videoex/ex8.pp

Program Example8;

{ Program to demonstrate the GetVideoModeCount function. }

Uses video, keyboard, vidutil;

Procedure DumpMode (M : TVideoMode; **Index** : Integer);

Var

S : **String**;

begin

```
Str(Index:2,S);
inc(Index);
TextOut(1,Index,'Data for mode '+S+' : ');
if M.Color then
  TextOut(19,Index,'  color,')
else
  TextOut(19,Index,'No color,');
Str(M.Row:3,S);
TextOut(28,Index,S+' rows ');
Str(M.Col:3,S);
TextOut(36,index,S+' columns ');
```

end;

Var

```
i,Count : Integer;
m : TVideoMode;
```

begin

```
InitVideo;
```

```
InitKeyboard;  
Count:=GetVideoModeCount;  
For I:=1 to Count do  
  begin  
    GetVideoModeData(I-1,M);  
    DumpMode(M, I-1);  
  end;  
TextOut(1,Count+1,'Press any key to exit');  
UpdateScreen(False);  
GetKeyEvent;  
DoneKeyboard;  
DoneVideo;  
end.
```

GetVideoModeData

Declaration: Function GetVideoModeData(Index : Word; Var Data: TVideoMode) : Boolean;

Description: GetVideoModeData returns the characteristics of the Index-th video mode in Data. Index is zero based, and has a maximum value of GetVideoModeCount-1. If the current driver does not support setting of modes (GetVideoModeCount=1) and Index is zero, the current mode is returned.

The function returns True if the mode data was retrieved successfully, False otherwise.

Errors: In case Index has a wrong value, False is returned.

See also: GetVideoModeCount ([508](#)), SetVideoMode ([511](#)), GetVideoMode ([507](#))

For an example, see GetVideoModeCount ([508](#)).

InitVideo

Declaration: procedure InitVideo;

Description: InitVideo Initializes the video subsystem. If the video system was already initialized, it does nothing. After the driver has been initialized, the VideoBuf and OldVideoBuf pointers are initialized, based on the ScreenWidth and ScreenHeight variables. When this is done, the screen is cleared.

Errors: if the driver fails to initialize, the ErrorCode variable is set.

See also: DoneVideo ([504](#))

For an example, see most other functions.

LockScreenUpdate

Declaration: Procedure LockScreenUpdate;

Description: LockScreenUpdate increments the screen update lock count with one. As long as the screen update lock count is not zero, UpdateScreen ([512](#)) will not actually update the screen.

This function can be used to optimize screen updating: If a lot of writing on the screen needs to be done (by possibly unknown functions), calling LockScreenUpdate before the drawing, and UnlockScreenUpdate ([512](#)) after the drawing, followed by a UpdateScreen ([512](#)) call, all writing will be shown on screen at once.

Errors: None.

See also: [UpdateScreen \(512\)](#), [UnlockScreenUpdate \(512\)](#), [GetLockScreenCount \(506\)](#)

For an example, see [GetLockScreenCount \(506\)](#).

SetCursorPos

Declaration: `procedure SetCursorPos(NewCursorX, NewCursorY: Word);`

Description: `SetCursorPos` positions the cursor on the given position: Column `NewCursorX` and row `NewCursorY`. The origin of the screen is the upper left corner, and has coordinates (0 , 0).

The current position is stored in the `CursorX` and `CursorY` variables.

Errors: None.

See also: [SetCursorType \(511\)](#)

Listing: `videoex/ex2.pp`

```
program example2;

uses video, keyboard;

Var
  P, PP, D : Integer;
  K: TKeyEvent;

  Procedure PutSquare (P : Integer; C : Char);

begin
  VideoBuf^[P]:=Ord(C)+($07 shr 8);
  VideoBuf^[P+ScreenWidth]:=Ord(c)+($07 shr 8);
  VideoBuf^[P+1]:=Ord(c)+($07 shr 8);
  VideoBuf^[P+ScreenWidth+1]:=Ord(c)+($07 shr 8);
end;

begin
  InitVideo;
  InitKeyBoard;
  P:=0;
  PP:=-1;
  Repeat
    If PP<>-1 then
      PutSquare(PP, ' ');
      PutSquare(P, '#');
      SetCursorPos(P Mod ScreenWidth, P div ScreenWidth);
      UpdateScreen(False);
      PP:=P;
    Repeat
      D:=0;
      K:=TranslateKeyEvent(GetKeyEvent);
      Case GetKeyEventCode(K) of
        kbdLeft : If (P Mod ScreenWidth)<>0 then
          D:=-1;
        kbdUp : If P>=ScreenWidth then
          D:=-ScreenWidth;
        kbdRight : If ((P+2) Mod ScreenWidth)<>0 then
```



```
        D:=1;
    kbdDown : if (P<(VideoBufSize div 2)-(ScreenWidth*2)) then
        D:=ScreenWidth;
    end;
    Until (D<>0) or (GetKeyEventChar(K)='q');
    P:=P+D;
    until GetKeyEventChar(K)='q';
    DoneKeyBoard;
    DoneVideo;
end.
```

SetCursorType

Declaration: `procedure SetCursorType(NewType: Word);`

Description: `SetCursorType` sets the cursor to the type specified in `NewType`.

crHidden the cursor is not visible.

crUnderLine the cursor is a small underline character (usually denoting insert mode).

crBlock the cursor is a block the size of a screen cell (usually denoting overwrite mode).

crHalfBlock the cursor is a block half the size of a screen cell.

Errors: None.

See also: `SetCursorPos` ([510](#))

SetVideoDriver

Declaration: `Function SetVideoDriver (Const Driver : TVideoDriver) : Boolean;`

Description: `SetVideoDriver` sets the videodriver to be used to `Driver`. If the current videodriver is initialized (after a call to `InitVideo`) then it does nothing and returns `False`.

A new driver can only be installed if the previous driver was not yet activated (i.e. before a call to `InitVideo` ([509](#))) or after it was deactivated (i.e after a call to `DoneVideo`).

For more information about installing a videodriver, see section [24.3](#), page [513](#).

Errors: If the current driver is initialized, then `False` is returned.

See also: The example video driver in section [24.3](#), page [513](#)

For an example, see the section on writing a custom video driver.

SetVideoMode

Declaration: `Function SetVideoMode(Mode: TVideoMode) : Boolean;`

Description: `SetVideoMode` sets the video mode to the mode specified in `Mode`:

```
TVideoMode = record
    Col,Row : Word;
    Color   : Boolean;
end;
```

If the call was succesful, then the screen will have `Col` columns and `Row` rows, and will be displaying in color if `Color` is `True`.

The function returns `True` if the mode was set succesfully, `False` otherwise.

Note that the video mode may not always be set. E.g. a console on Linux or a telnet session cannot always set the mode. It is important to check the error value returned by this function if it was not succesful.

The mode can be set when the video driver has not yet been initialized (i.e. before `InitVideo` (509) was called) In that case, the video mode will be stored, and after the driver was initialized, an attempt will be made to set the requested mode. Changing the video driver before the call to `InitVideo` will clear the stored video mode.

To know which modes are valid, use the `GetVideoModeCount` (508) and `GetVideoModeData` (509) functions. To retrieve the current video mode, use the `GetVideoMode` (507) procedure.

Errors: If the specified mode cannot be set, then `errVioNoSuchMode` may be set in `ErrorCode`

See also: `GetVideoModeCount` (508) `GetVideoModeData` (509) `GetVideoMode` (507)

UnlockScreenUpdate

Declaration: `Procedure UnlockScreenUpdate;`

Description: `UnlockScreenUpdate` decrements the screen update lock count with one if it is larger than zero. When the lock count reaches zero, the `UpdateScreen` (512) will actually update the screen. No screen update will be performed as long as the screen update lock count is nonzero. This mechanism can be used to increase screen performance in case a lot of writing is done.

It is important to make sure that each call to `LockScreenUpdate` (509) is matched by exactly one call to `UnlockScreenUpdate`

Errors: None.

See also: `LockScreenUpdate` (509), `GetLockScreenCount` (506), `UpdateScreen` (512)

For an example, see `GetLockScreenCount` (506).

UpdateScreen

Declaration: `procedure UpdateScreen(Force: Boolean);`

Description: `UpdateScreen` synchronizes the actual screen with the contents of the `VideoBuf` internal buffer. The parameter `Force` specifies whether the whole screen has to be redrawn (`Force=True`) or only parts that have changed since the last update of the screen.

The `Video` unit keeps an internal copy of the screen as it last wrote it to the screen (in the `OldVideoBuf` array). The current contents of `VideoBuf` are examined to see what locations on the screen need to be updated. On slow terminals (e.g. a LINUX telnet session) this mechanism can speed up the screen redraw considerably.

Errors: None.

See also: `ClearScreen` (503)

For an example, see most other functions.

24.3 Writing a custom video driver

Writing a custom video driver is not difficult, and generally means implementing a couple of functions, which should be registered with the **SetVideoDriver** (511) function. The various functions that can be implemented are located in the **TVideoDriver** record:

```
TVideoDriver = Record
  InitDriver      : Procedure;
  DoneDriver      : Procedure;
  UpdateScreen    : Procedure(Force : Boolean);
  ClearScreen     : Procedure;
  SetVideoMode    : Function (Const Mode : TVideoMode) : Boolean;
  GetVideoModeCount : Function : Word;
  GetVideoModeData : Function(Index : Word; Var Data : TVideoMode) : Boolean;
  SetCursorPos    : procedure (NewCursorX, NewCursorY: Word);
  GetCursorType   : function : Word;
  SetCursorType   : procedure (NewType: Word);
  GetCapabilities : Function : Word;
end;
```

Not all of these functions must be implemented. In fact, the only absolutely necessary function to write a functioning driver is the **UpdateScreen** function. The general calls in the **Video** unit will check which functionality is implemented by the driver.

The functionality of these calls is the same as the functionality of the calls in the **video** unit, so the expected behaviour can be found in the previous section. Some of the calls, however, need some additional remarks.

InitDriver Called by **InitVideo**, this function should initialize any data structures needed for the functionality of the driver, maybe do some screen initializations. The function is guaranteed to be called only once; It can only be called again after a call to **DoneVideo**. The variables **ScreenWidth** and **ScreenHeight** should be initialized correctly after a call to this function, as the **InitVideo** call will initialize the **VideoBuf** and **OldVideoBuf** arrays based on their values.

DoneDriver This should clean up any structures that have been initialized in the **InitDriver** function. It should possibly also restore the screen as it was before the driver was initialized. The **VideoBuf** and **OldVideoBuf** arrays will be disposed of by the general **DoneVideo** call.

UpdateScreen This is the only required function of the driver. It should update the screen based on the **VideoBuf** array's contents. It can optimize this process by comparing the values with values in the **OldVideoBuf** array. After updating the screen, the **UpdateScreen** procedure should update the **OldVideoBuf** by itself. If the **Force** parameter is **True**, the whole screen should be updated, not just the changed values.

ClearScreen If there is a faster way to clear the screen than to write spaces in all character cells, then it can be implemented here. If the driver does not implement this function, then the general routines will write spaces in all video cells, and will call **UpdateScreen(True)**.

SetVideoMode Should set the desired video mode, if available. It should return **True** if the mode was set, **False** if not.

GetVideoModeCount Should return the number of supported video modes. If no modes are supported, this function should not be implemented; the general routines will return 1. (for the current mode)

GetVideoModeData Should return the data for the `Index`-th mode; `Index` is zero based. The function should return true if the data was returned correctly, false if `Index` contains an invalid index. If this is not implemented, then the general routine will return the current video mode when `Index` equals 0.

GetCapabilities If this function is not implemented, zero (i.e. no capabilities) will be returned by the general function.

The following unit shows how to override a video driver, with a driver that writes debug information to a file.

Listing: videoex/viddbg.pp

```
unit viddbg;
```

```
Interface
```

```
uses video;
```

```
Procedure StartVideoLogging;
```

```
Procedure StopVideoLogging;
```

```
Function IsVideoLogging : Boolean;
```

```
Procedure SetVideoLogFileName (FileName : String);
```

```
Const
```

```
    DetailedVideoLogging : Boolean = False;
```

```
Implementation
```

```
uses sysutils, keyboard;
```

```
var
```

```
    NewVideoDriver,  
    OldVideoDriver : TVideoDriver;  
    Active, Logging : Boolean;  
    LogFileName : String;  
    VideoLog : Text;
```

```
Function TimeStamp : String;
```

```
begin
```

```
    TimeStamp := FormatDateTime('hh:nn:ss', Time());
```

```
end;
```

```
Procedure StartVideoLogging;
```

```
begin
```

```
    Logging := True;
```

```
    WriteLn(VideoLog, 'Start logging video operations at: ', TimeStamp);
```

```
end;
```

```
Procedure StopVideoLogging;
```

```
begin
```

```
    WriteLn(VideoLog, 'Stop logging video operations at: ', TimeStamp);
```

```
    Logging := False;
```

```
end;
```

```
Function IsVideoLogging : Boolean;

begin
  IsVideoLogging:=Logging;
end;

Var
  ColUpd,RowUpd : Array[0..1024] of Integer;

Procedure DumpScreenStatistics(Force : Boolean);

Var
  I,Count : Integer;

begin
  If Force then
    Write(VideoLog,'forced ');
    WriteLn(VideoLog,'video update at ',TimeStamp,' : ');
    FillChar(ColUpd,SizeOf(ColUpd),#0);
    FillChar(RowUpd,SizeOf(RowUpd),#0);
    Count:=0;
    For I:=0 to VideoBufSize div SizeOf(TVideoCell) do
      begin
        If VideoBuf^[i]<>OldVideoBuf^[i] then
          begin
            Inc(Count);
            Inc(ColUpd[I mod ScreenWidth]);
            Inc(RowUpd[I div ScreenHeight]);
          end;
        end;
      Write(VideoLog,Count,' videocells differed divided over ');
      Count:=0;
      For I:=0 to ScreenWidth-1 do
        If ColUpd[I]<>0 then
          Inc(Count);
      Write(VideoLog,Count,' columns and ');
      Count:=0;
      For I:=0 to ScreenHeight-1 do
        If RowUpd[I]<>0 then
          Inc(Count);
      WriteLn(VideoLog,Count,' rows. ');
      If DetailedVideoLogging Then
        begin
          For I:=0 to ScreenWidth-1 do
            If (ColUpd[I]<>0) then
              WriteLn(VideoLog,'Col ',i,' : ',ColUpd[I]:3,' rows changed');
          For I:=0 to ScreenHeight-1 do
            If (RowUpd[I]<>0) then
              WriteLn(VideoLog,'Row ',i,' : ',RowUpd[I]:3,' columns changed');
          end;
        end;
    end;

Procedure LogUpdateScreen(Force : Boolean);

begin
  If Logging then
    DumpScreenStatistics(Force);
    OldVideoDriver.UpdateScreen(Force);
```

```
end;  
  
Procedure LogInitVideo;  
  
begin  
    OldVideoDriver.InitDriver();  
    Assign (VideoLog, logFileName);  
    Rewrite (VideoLog);  
    Active:=True;  
    StartVideoLogging;  
end;  
  
Procedure LogDoneVideo;  
  
begin  
    StopVideoLogging;  
    Close (VideoLog);  
    Active:=False;  
    OldVideoDriver.DoneDriver();  
end;  
  
Procedure SetVideoLogFileName (FileName : String);  
  
begin  
    If Not Active then  
        LogFileName:=FileName;  
    end;  
  
Initialization  
    GetVideoDriver (OldVideoDriver);  
    NewVideoDriver:=OldVideoDriver;  
    NewVideoDriver.UpdateScreen:=@LogUpdateScreen;  
    NewVideoDriver.InitDriver:=@LogInitVideo;  
    NewVideoDriver.DoneDriver:=@LogDoneVideo;  
    LogFileName:=' Video.log';  
    Logging:=False;  
    SetVideoDriver (NewVideoDriver);  
end.
```

The unit can be used in any of the demonstration programs, by simply including it in the uses clause. Setting DetailedVideoLogging to True will create a more detailed log (but will also slow down functioning)

Index

Abstract, 315
Accept, 376–378
Access, 189
AddDisk, 45
AddDisk (Linux only), 419
AdjustLineBreaks, 439
Alarm, 190
allocate_ldt_descriptors, 85
allocate_memory_block, 87
AnsiCompareStr, 439
AnsiCompareText, 440
AnsiExtractQuotedStr, 441
AnsiLastChar, 442
AnsiLowerCase, 442
AnsiQuotedStr, 443
AnsiStrComp, 443
AnsiStrIComp, 444
AnsiStrLastChar, 445
AnsiStrLComp, 445
AnsiStrLIComp, 446
AnsiStrLower, 447
AnsiStrUpper, 448
AnsiUpperCase, 448
AppendStr, 449
Arc, 118
arccos, 259
arcosh, 259
arcsin, 260
arctan2, 261
arsinh, 261
artanh, 262
AssignCrt, 29
AssignLst, 373
AssignPipe, 190
AssignStr, 449
AssignStream, 191

Bar, 118
Bar3D, 118
BaseName, 193
BCDToInt, 450
Bind, 378, 379

ceil, 262
CFMakeRaw, 193

CFSetISpeed, 193
CFSetOSpeed, 194
ChangeFileExt, 422
Chmod, 195
Chown, 194
Circle, 119
ClearDevice, 119
ClearScreen, 503
ClearViewPort, 119
Clone, 196
CloseDir, 198
CloseGraph, 119
ClrEol, 30
ClrScr, 30
CompareMem, 450
CompareStr, 451
CompareText, 451
Connect, 379, 380
copyfromdos, 87
copytodos, 87
cosh, 263
cotan, 263
create_code_segment_alias_descriptor, 88
CreateDir, 419
CreateShellArgV, 198
CursorBig, 30
CursorOff, 31
CursorOn, 31
cycletorad, 263

Date, 405
DateTimeToFileDate, 406
DateTimeToStr, 406
DateTimeToString, 407
DateTimeToSystemTime, 408
DateTimeToTimeStamp, 408
DateToStr, 409
DayOfWeek, 409
DecodeDate, 409
DecodeTime, 410
DefaultErrorHandler, 504
degtograd, 264
degtorad, 264
Delay, 31
DeleteFile, 422

DelLine, 32
DetectGraph, 119
DetectMouse, 291
DirName, 199
disable, 88
DiskFree, 45, 420
DiskSize, 46, 420
DisposeStr, 315, 452
DoDirSeparators, 423
DoneKeyboard, 162
DoneMouse, 292
DoneVideo, 504
DosExitCode, 46
dosmemfillchar, 88
dosmemfillword, 89
dosmemget, 89
dosmemmove, 90
dosmempu, 90
DosVersion, 47
DrawPoly, 120
DumpHeap, 138
Dup, 199
Dup2, 200
dxe_load, 61

Ellipse, 120
Emms, 289
enable, 90
EncodeDate, 410
EncodeTime, 411
EnvCount, 48
EnvStr, 48
EpochToLocal, 201
Exec, 48
Execl, 201
Execl, 202
Execlp, 203
Execv, 203
Execve, 204
Execvp, 205
ExpandFileName, 423
ExpandUNCFileName, 424
ExtractFileDir, 424
ExtractFileDrive, 425
ExtractFileExt, 425
ExtractFileName, 425
ExtractFilePath, 426
ExtractRelativePath, 426

Fcntl, 215
FD_Clr, 206
FD_IsSet, 206
FD_Set, 207
FD_ZERO, 206

fdClose, 207
fdFlush, 207
fdOpen, 207
fdRead, 208
fdSeek, 210
fdTruncate, 210
fdWrite, 210
FExpand, 48, 210
FileAge, 427
FileClose, 427
FileCreate, 427
FileDateToDateTime, 411
FileExists, 428
FileGetAttr, 429
FileGetDate, 430
FileOpen, 430
FileRead, 431
FileSearch, 431
FileSeek, 432
FileSetAttr (Not on Linux), 432
FileSetDate (Not on Linux), 433
FileTruncate, 433
FileWrite, 433
FillEllipse, 120
FillPoly, 120
FindClose, 49, 433
FindFirst, 49, 434
FindNext, 50, 434
FindPropInfo, 477
FloatToStr, 453
FloatToStrF, 453
FloatToText, 455
FLock, 211
FloodFill, 120
floor, 265
FmtStr, 456
FNMatch, 211
Fork, 216
Format, 456
FormatBuf, 461
FormatDateTime, 412
free_ltd_descriptor, 91
free_memory_block, 91
free_rm_callback, 91
FRename, 216
frexp, 265
FSearch, 50, 212
FSplit, 51, 212
FSSStat, 213
FStat, 214
ftok, 146
FunctionKeyName, 162

get_cs, 92

get_descriptor_access_rights, 92
get_ds, 92
get_linear_addr, 92
get_meminfo, 93
get_next_selector_increment_value, 94
get_page_size, 94
get_pm_interrupt, 94
get_rm_callback, 95
get_rm_interrupt, 97
get_run_mode, 98
get_segment_base_address, 98
get_segment_limit, 99
get_ss, 99
GetArcCoords, 121
GetAspectRatio, 121
GetBkColor, 121
GetCapabilities, 504
GetCBreak, 51
GetColor, 121
GetCurrentDir, 421
GetCursorType, 505
GetDate, 52, 217
GetDateTime, 217
GetDefaultPalette, 121
GetDirs, 435
GetDomainName, 217
GetDriverName, 121
GetEGid, 218
GetEnumName, 478
GetEnumProp, 478
GetEnumValue, 479
GetEnv, 52, 219
GetEpochTime, 219
GetEUid, 218
GetFAttr, 53
GetFillPattern, 122
GetFillSettings, 122
GetFloatProp, 479
GetFS, 219
GetFTime, 53
GetGid, 220
GetGraphMode, 122
GetHostName, 220
GetImage, 122
GetInt64Prop, 480
GetIntVec, 54
GetKeyboardDriver, 163
GetKeyEvent, 163
GetKeyEventChar, 164
GetKeyEventCode, 164
GetKeyEventFlags, 165
GetKeyEventShiftState, 165
GetKeyEventUnicode, 166
GetLastButtonPress, 301
GetLastButtonRelease, 302
GetLineSettings, 122
GetLocalTimezone, 221
GetLockScreenCount, 506
GetLongName, 54
GetLongOpts, 65
GetMaxColor, 122
GetMaxMode, 123
GetMaxX, 123
GetMaxY, 123
GetMethodProp, 481
GetModeName, 123
GetModeRange, 123
GetMouseButtons, 292
GetMouseDriver, 293
GetMouseEvent, 293
GetMouseState, 303
GetMouseX, 293
GetMouseY, 294
GetObjectProp, 483
GetObjectPropClass, 484
Getopt, 65
GetOrdProp, 484
GetPalette, 124
GetPaletteSize, 124
GetPeerName, 381
GetPid, 221
GetPixel, 124
GetPPid, 221
GetPriority, 222
GetPropInfo, 485
GetPropInfos, 486
GetPropList, 487
GetPropValue, 488
GetSetProp, 488
GetShortName, 55
GetSocketName, 381
GetSocketOptions, 382
GetStrProp, 489
GetTextSettings, 124
GetTime, 55, 222
GetTimeOfDay, 223
GetTimezoneFile, 223
GetTypeData, 490
GetUid, 224
GetVariantProp, 490
GetVerify, 56
GetVideoDriver, 507
GetVideoMode, 507
GetVideoModeCount, 508
GetVideoModeData, 509
GetViewSettings, 124
GetX, 124
GetY, 125

Glob, 224
global_dos_alloc, 99
global_dos_free, 101
GlobFree, 225
GotoXY, 32
Gpm_AnyDouble, 70
Gpm_AnySingle, 71
Gpm_AnyTriple, 71
Gpm_Close, 71
Gpm_FitValues, 71
Gpm_FitValuesM, 71
Gpm_GetEvent, 72
Gpm_GetLibVersion, 73
Gpm_GetServerVersion, 73
Gpm_GetSnapshot, 73
Gpm_LowerRoi, 73
Gpm_Open, 74
Gpm_PopRoi, 74
Gpm_PushRoi, 74
Gpm_RaiseRoi, 74
Gpm_Repeat, 75
Gpm_StrictDouble, 75
Gpm_StrictSingle, 75
Gpm_StrictTriple, 75
gradtodeg, 266
gradtorad, 266
GraphDefaults, 125
GraphErrorMsg, 125
GraphResult, 125

HideMouse, 294, 304
HighVideo, 33
hypot, 267

ImageSize, 126
IncMonth, 412
InitGraph, 126
InitKeyBoard, 166
InitMouse, 295, 304
InitVideo, 509
inportb, 101
inportl, 101
inportw, 101
InsLine, 33
InstallUserDriver, 126
InstallUserFont, 127
intpower, 267
Intr, 56
IntToHex, 462
IntToStr, 462
IOctl, 225
IOperm, 225
IsATTY, 226
IsFunctionKey, 166

IsLeapYear, 413
IsPublishedProp, 490
IsStoredProp, 491
IsValidIdent, 463

Keep, 56
KeyEventToString, 167
KeyPressed, 34
Kill, 228

LastDelimiter, 464
ldexp, 268
LeftStr, 464
Line, 127
LineRel, 127
LineTo, 127
Link, 229
Listen, 382
lnxp1, 268
LoadStr, 464
LocalToEpoch, 230
lock_code, 102
lock_data, 102
lock_linear_region, 102
LockScreenUpdate, 509
log10, 269
log2, 269
logn, 270
LongDiv, 317
LongMul, 317
LowerCase, 465
LowVideo, 34
LPressed, 305
LStat, 228

MarkHeap, 138
max, 270
maxIntValue, 271
maxvalue, 271
mean, 272
meanandstddev, 273
min, 273
minIntValue, 274
minvalue, 275
MkFifo, 231
MMap, 231
momentskewkurtosis, 275
MoveRel, 127
MoveTo, 128
MPressed, 305
MSDos, 56
MSecsToTimeStamp, 414
msgctl, 147
msgget, 146

msgrcv, 147
msgsnd, 146
MUnMap, 233

NewStr, 314, 465
Nice, 233
norm, 276
NormVideo, 35
NoSound, 35
Now, 414
npsetup, 63

Octal, 234
OpenDir, 234
outportb, 103
outportl, 103
outportw, 103
OutText, 128
OutTextXY, 128

PackTime, 57
pause, 235
PClose, 235
PieSlice, 128
PollKeyEvent, 167
PollMouseEvent, 295
PollShiftStateEvent, 168
POpen, 236
popnstddev, 277
popnvariance, 277
power, 278
PropIsType, 492
PropType, 493
PutImage, 128
PutKeyEvent, 168
PutMouseEvent, 295
PutPixel, 129

QuotedStr, 465

radtocycle, 279
rattodeg, 279
rattograd, 280
randg, 280
ReadDir, 236
ReadKey, 35
ReadLink, 237
ReadPort, 238
ReadPortB, 238
ReadPortL, 238
ReadPortW, 239
ReadTimezoneFile, 239
realintr, 104
Rectangle, 129
Recv, 383

RegisterBGIDriver, 129
RegisterBGIFont, 129
RegisterObjects, 315
RegisterType, 315
RemoveDir, 421
RenameFile, 435
RestoreCRTMode, 129
RightStr, 466
RPressed, 305

S_ISBLK, 226
S_ISCHR, 226
S_ISDIR, 226
S_ISFIFO, 226
S_ISLNK, 227
S_ISREG, 227
S_ISSOCK, 227
Sector, 130
SeekDir, 239
seg_fillchar, 104
seg_fillword, 105
seg_move, 106
segment_to_descriptor, 105
Select, 240
SelectText, 241
semctl, 151
semget, 150
semop, 150
Send, 383
set_descriptor_access_rights, 106
set_pm_interrupt, 106
set_rm_interrupt, 107
set_segment_base_address, 108
set_segment_limit, 108
SetActivePage, 130
SetAllPalette, 130
SetAspectRatio, 130
SetBkColor, 130
SetCBreak, 57
SetColor, 131
SetCurrentDir, 422
SetCursorPos, 510
SetCursorType, 511
SetDate, 58
SetDirSeparators, 436
SetEnumProp, 494
SetExtraInfo, 139
SetFAttr, 58
SetFillPattern, 131
SetFillStyle, 131
SetFloatProp, 494
SetFTime, 58
SetGraphBufSize, 131
SetGraphMode, 132

SetHeapTraceOutput, 140
SetInt64Prop, 494
SetIntVec, 59
SetKeyboardDriver, 169
SetLineStyle, 132
SetMethodProp, 495
SetMouseAscii, 305
SetMouseDriver, 296
SetMouseHideWindow, 306
SetMousePos, 307
SetMouseShape, 308
SetMouseSpeed, 309
SetMouseWindow, 310
SetMouseXRange, 310
SetMouseXY, 296
SetMouseYRange, 311
SetObjectProp, 495
SetOrdProp, 495
SetPalette, 132
SetPriority, 241
SetPropValue, 496
SetRGBPalette, 132
SetSetProp, 496
SetSocketOptions, 384
SetStrProp, 497
SetTextJustify, 133
SetTextStyle, 133
SetTime, 59
SetToString, 497
SetUserCharSize, 133
SetVariantProp, 498
SetVerify, 59
SetVideoDriver, 511
SetVideoMode, 511
SetViewPort, 134
SetVisualPage, 134
SetWriteMode, 134
Shell, 241
ShiftStateToString, 170
shmat, 156
shmctl, 156
shmdt, 156
shmget, 155
ShowMouse, 296, 311
Shutdown, 384
SigAction, 242
Signal, 245
SigPending, 243
SigProcMask, 243
SigRaise, 244
SigSuspend, 244
sincos, 281
sinh, 281
Sock2File, 384
Sock2Text, 385
Socket, 385
SocketPair, 385
Sound, 36
stddev, 282
Str2UnixSockAddr, 385
StrAlloc, 386, 437
StrBufSize, 437
StrCat, 386
StrComp, 387
StrCopy, 387
StrDispose, 388, 438
StrECopy, 388
StrEnd, 389
StrFmt, 466
StrIComp, 389
StringToPPchar, 245
StringToSet, 498
StrLCat, 390
StrLComp, 390
StrLCopy, 391
StrLen, 391
StrLFmt, 467
StrLIComp, 392
StrLower, 392
StrMove, 392
StrNew, 393
StrPas, 393, 439
StrPCopy, 394, 438
StrPLCopy, 438
StrPos, 394
StrRScan, 395
StrScan, 395
StrToDate, 415
StrToDateTime, 415
StrToInt, 467
StrToIntDef, 468
StrToTime, 416
StrUpper, 395
sum, 282
sumofsquares, 283
sumsandsquares, 284
SwapVectors, 59
SymLink, 246
SysInfo, 247
SystemTimeToDateTime, 417

tan, 284
tanh, 285
tb_size, 108
TBufStream.Close, 337
TBufStream.Done, 336
TBufStream.Flush, 337
TBufStream.Init, 336

-
- TBufStream.Open, 338
 - TBufStream.Read, 338
 - TBufStream.Seek, 338
 - TBufStream.Truncate, 338
 - TBufStream.Write, 339
 - TCDrain, 248
 - TCFlow, 248
 - TCFlush, 249
 - TCGetAttr, 249
 - TCGetPGrp, 250
 - TCollection.At, 343
 - TCollection.AtDelete, 351
 - TCollection.AtFree, 350
 - TCollection.AtInsert, 353
 - TCollection.AtPut, 353
 - TCollection.Delete, 349
 - TCollection.DeleteAll, 348
 - TCollection.Done, 343
 - TCollection.Error, 353
 - TCollection.FirstThat, 345
 - TCollection.ForEach, 352
 - TCollection.Free, 348
 - TCollection.FreeAll, 347
 - TCollection.FreeItem, 351
 - TCollection.GetItem, 344
 - TCollection.IndexOf, 344
 - TCollection.Init, 342
 - TCollection.Insert, 349
 - TCollection.LastThat, 345
 - TCollection.Load, 342
 - TCollection.Pack, 346
 - TCollection.PutItem, 354
 - TCollection.SetLimit, 353
 - TCollection.Store, 354
 - TCSendBreak, 250
 - TCSetAttr, 250
 - TCSetPGrp, 251
 - TDateTime, 405
 - TDosStream.Close, 332
 - TDosStream.Done, 332
 - TDosStream.Init, 332
 - TDosStream.Open, 334
 - TDosStream.Read, 335
 - TDosStream.Seek, 333
 - TDosStream.Truncate, 333
 - TDosStream.Write, 335
 - TellDir, 251
 - TextBackground, 36
 - TextColor, 37
 - TextHeight, 134
 - TextMode, 37
 - TextWidth, 134
 - Time, 417
 - TimeStampToDateTime, 418
 - TimeStampToMSecs, 418
 - TimeToStr, 418
 - TMemoryStream.Done, 340
 - TMemoryStream.Init, 339
 - TMemoryStream.Read, 340
 - TMemoryStream.Truncate, 340
 - TMemoryStream.Write, 341
 - TObject.Done, 323
 - TObject.Free, 322
 - TObject.Init, 322
 - totalvariance, 285
 - transfer_buffer, 109
 - TranslateKeyEvent, 170
 - TranslateKeyEventUnicode, 170
 - TRect.Assign, 322
 - TRect.Contains, 319
 - TRect.Copy, 319
 - TRect.Empty, 318
 - TRect.Equals, 318
 - TRect.Grow, 321
 - TRect.Intersect, 320
 - TRect.Move, 321
 - TRect.Union, 319
 - TResourceCollection.FreeItem, 365
 - TResourceCollection.GetItem, 365
 - TResourceCollection.KeyOf, 365
 - TResourceCollection.PutItem, 365
 - TResourceFile.Count, 367
 - TResourceFile.Delete, 368
 - TResourceFile.Done, 366
 - TResourceFile.Flush, 367
 - TResourceFile.Get, 367
 - TResourceFile.Init, 366
 - TResourceFile.KeyAt, 367
 - TResourceFile.Put, 368
 - TResourceFile.SwitchTo, 367
 - Trim, 468
 - TrimLeft, 469
 - TrimRight, 470
 - TSortedCollection.Compare, 357
 - TSortedCollection.IndexOf, 356
 - TSortedCollection.Init, 356
 - TSortedCollection.Insert, 358
 - TSortedCollection.KeyOf, 356
 - TSortedCollection.Load, 356
 - TSortedCollection.Search, 357
 - TSortedCollection.Store, 359
 - TStrCollection.Compare, 362
 - TStrCollection.FreeItem, 363
 - TStrCollection.GetItem, 362
 - TStrCollection.PutItem, 363
 - TStream.Close, 327
 - TStream.CopyFrom, 330
 - TStream.Error, 329

TStream.Flush, 328
TStream.Get, 324
TStream.GetPos, 325
TStream.GetSize, 326
TStream.Open, 327
TStream.Put, 328
TStream.Read, 330
TStream.ReadStr, 326
TStream.Reset, 328
TStream.Seek, 329
TStream.StrRead, 325
TStream.StrWrite, 329
TStream.Truncate, 328
TStream.Write, 330
TStream.WriteStr, 329
TStringCollection.Compare, 360
TStringCollection.FreeItem, 361
TStringCollection.GetItem, 360
TStringCollection.PutItem, 361
TStringList.Done, 369
TStringList.Get, 369
TStringList.Load, 368
TStrListMaker.Done, 370
TStrListMaker.Init, 369
TStrListMaker.Put, 370
TStrListMaker.Store, 370
TTYName, 251
TUnSortedStrCollection.Insert, 363

Umask, 251
Uname, 252
UnLink, 252
unlock_code, 109
unlock_data, 109
unlock_linear_region, 109
UnlockScreenUpdate, 512
UnPackTime, 60
UpdateScreen, 512
UpperCase, 470
Utime, 252

variance, 286

WaitPid, 253
WhereX, 37
WhereY, 38
Window, 38
WritePort, 254
WritePortB, 254
WritePortL, 254
WritePortW, 255