Free Pascal supplied units : Reference guide.

Reference guide for standard Free Pascal units.
1.3
February 1999

Michaël Van Canneyt
Florian Klämpfl

Contents

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 gave 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 `typewriter` 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 and LINUX. The unit was first written for DOS by Florian klämpfl. The unit was ported to LINUX by Mark May[1], and enhanced by Michaël Van Canneyt 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 an 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

[1]Current e-mail address mmay@dnaco.net

```
  TextAttr: Byte = $07;
  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: 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 (27)

```
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.
```

BigCursor

Declaration: `Procedure BigCursor ;`

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

Errors: None.

See also: CursorOn (20), CursorOff (20)

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 (21), InsLine (22), ClrScr (20)

```
Program Example9;
uses Crt;

{ Program to demonstrate the ClrEol function. }

begin
  Write('This line will be cleared from the',
        ' cursor postion 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 (27)

```
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 (20), BigCursor (19)

CursorOn

Declaration: `Procedure CursorOn ;`

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

Errors: None.

See also: BigCursor (19), CursorOff (20)

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 (25), NoSound (24)

```
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 (19), InsLine (22), ClrScr (20)

```
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 (26), WhereY (27), Window (27)

```
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 (26), TextBackground (25), LowVideo (23), NormVideo (24)

```
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 (19), DelLine (21), ClrScr (20)

```
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 (24)

```
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 (26), TextBackground (25), HighVideo (22), NormVideo (24)

For an example, see HighVideo (22)

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 (26), TextBackground (25), LowVideo (23), HighVideo (22)

For an example, see HighVideo (22)

NoSound

Declaration: `Procedure NoSound ;`

Description: Stops the speaker sound. This is not supported in LINUX

Errors: None.

See also: Sound (25)

```
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 (23)

```
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`. This is not supported in LINUX

Errors: None.

See also: NoSound (24)

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 (26), HighVideo (22), LowVideo (23), NormVideo (24)

```
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 (25), HighVideo (22), LowVideo (23), NormVideo (24)

```
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.
```

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 (21), WhereY (27), Window (27)

```
Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  Writeln('Cursor postion: 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 (21), WhereX (26), Window (27)

```
Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  Writeln('Cursor postion: 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 (21), WhereX (26), WhereY (27), ClrScr (20)

```
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.
```

ScrollWindow

Declaration: `Procedure ScrollWindow (X1,Y1,X2,Y2 : Byte; Count : Longint);`

Description: ScrollWindow scrolls the contents of the window defined by the upper-left

Errors: None.

See also: Window (27), ClrScr (20)

SaveScreenRegion

Declaration: `Function SaveScreenRegion (X1,Y1,X2,Y2, var P : pointer) : Boolean;`

Description: SaveScreenRegion writes the the contents of the window defined by the upper-left

Errors: None.

See also: RestoreScreenRegion (28), Window (27)

RestoreScreenRegion

Declaration: `Procedure RestoreScreenRegion (X1,Y1,X2,Y2, var P : pointer);`

Description: SaveScreenRegion writes the the contents of the memory location pointed to

Errors: None

See also: SaveScreenRegion (28), Window (27)

Chapter 2

The DOS unit.

This chapter describes the DOS unit for Free pascal, both under DOS and LINUX. The unit was first written for DOS by Florian klämpfl. The unit was ported to LINUX by Mark May[1], and enhanced by Michaël Van Canneyt. Under LINUX, some of the functionality is lost, as it is either impossible or meaningless to implement it. Other than that, the functionality for both operating systems is the same. 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 DOS unit.

2.1 Types, Variables, Constants

Constants

The DOS unit implements the following constants:

```
{Bitmasks for CPU Flags}
fcarry =     $0001;
fparity =    $0004;
fauxiliary = $0010;
fzero =      $0040;
fsign =      $0080;
foverflow  = $0800;
{Bitmasks for file attribute}
readonly =   $01;
hidden =     $02;
sysfile =    $04;
volumeid =   $08;
directory =  $10;
archive =    $20;
anyfile =    $3F;
fmclosed =   $D7B0;
fminput =    $D7B1;
fmoutput =   $D7B2;
fminout =    $D7B3;
```

[1]Current e-mail address mmay@dnaco.net

Types

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

```
  ComStr  = String[127]; { For command-lines }
  PathStr = String[79];  { For full path for file names }
  DirStr  = String[67];  { For Directory and (DOS) drive string }
  NameStr = String[8];   { For Name of file }
  ExtStr  = String[4];   { For Extension of file }
```

Under LINUX, these strings all have length 255.

```
{$PACKRECORDS 1}
  SearchRec = 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 meachanism 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,
             f51,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 (43) and UnPackTime (45) for setting/reading file times with GetFTime (40) and SetFTime (44).

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 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 LINUX only. This list is used to determine which disks to use in the DiskFree (32) and DiskSize (33) calls. The DiskFree (32) and DiskSize (33) 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.
- `'/fd1/.'` for the second floppy-drive.
- `'/'` 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 (32), DiskSize (33)

DiskFree

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

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. Typically, the free space is the size of a disk block, multiplied by the number of free blocks on the disk.
For linux 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.
- `'/fd1/.'` for the second floppy-drive.
- `'/'` for the first hard disk.

There is room for 1-26 drives. You can add a drive with the AddDisk (32) procedure. These settings can be coded in `dos.pp`, in the initialization part.

Errors: -1 when a failure occurs, or an invalid `drivenr` is given.

See also: DiskSize (33), AddDisk (32)

```
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) : longint;`

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 linux 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.
- `'/fd1/.'` for the second floppy-drive.
- `'/'` for the first hard disk.

There is room for 1-26 drives. You can add a drive with the AddDisk (32) 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 (32), AddDisk (32)

For an example, see DiskFree (32).

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 (35)

```
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 ',DosExitCode);
end.
```

DosVersion

Declaration: `Function DosVersion : Word;`

Description: `DosVersion` returns the DOS version number. On LINUX systems, it returns the Linux version (The first 2 numbers, e.g Linux version 2.1.76 will give you DosVersion 2.1)

Errors: None.

See also:

```
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 (35), GetEnv (159)

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. This may cause problems under LINUX. The LINUX unit solves this problem.

See also: EnvCount (34), GetEnv (159)

```
Program Example13;
uses Dos;

{ Program to demonstrate the EnvCount and EnvStr function. }

var
  i : Longint;
begin
  WriteLn('Current Environment is:');
  for i:=1to 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 (33)

For an example, see DosExitCode (33)

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 (under DOS). The resulting name is converted to uppercase on DOS systems. Under LINUX, the name is left as it is. (filenames are case sensitive under Unix)

Errors: FSplit (38)

```
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 ',DosExitCode);
end.
```

See also:

FindClose

Declaration: `Procedure FindClose (Var F: SearchRec);`

Description: **linux only** Under LINUX, the `findfirst/findnext` calls have to be mimicked. An internal table of file descriptors is kept. When using different `searchrecs` at the same time, the system may run out of file descriptors for directories. 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. It is recommended to use the LINUX call `Glob` when looking for files.

Errors: None.

See also: Glob (163).

FindFirst

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

Description: `FindFirst` searches the file specified in `Path`, checks the atrributes specified in `Attr`. 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.

Errors: Errors are reported in DosError.

See also: FindNext (37), FindClose (36)

```
Program Example7;
uses Dos;

{ Program to demonstrate the FindFirst and FindNext function. }

var
  Dir : SearchRec;
begin
```

```
  FindFirst ('*.*',$20,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 (36), FindClose (36)

For an example, see FindFirst (36).

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 (or colons under LINUX). When no file is found, an empty string is returned.

Errors: None.

See also: FExpand (35)

```
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`.) Under LINUX, the extension is taken to be all letters after the last dot (.).

Errors: None.

See also: FSearch (37)

```
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. 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.

Errors: Under Linux, this exists but is not implemented, i.e. the call does nothing.

See also: SetCBreak (43)

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 (41), SetDate (44)

```
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`. Under LINUX, case is important when looking for `EnvVar`. When there is no environment variable `EnvVar` defined, an empty string is returned.

Errors: None.

See also: EnvCount (34), EnvStr (35)

```
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 = 01h`
- `Hidden = 02h`
- `SysFile = 04h`
- `VolumeId = 08h`
- `Directory = 10h`
- `Archive = 20h`
- `AnyFile = 3fh`

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 (44)

```
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 $20)<>0 then WriteLn('- Archive');
  if (Attr and $10)<>0 then WriteLn('- Directory');
  if (Attr and $4)<>0 then WriteLn('- Read-Only');
  if (Attr and $2)<>0 then WriteLn('- System');
  if (Attr and $1)<>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 (44), PackTime (43), UnPackTime (45)

```
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`.

Errors: None. Under LINUX, this call exists bout isn't implemented, i.e. it does nothing.

See also: SetIntVec (44)

GetTime

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

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

Errors: None.

See also: GetDate (38), SetTime (45)

```
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.

Errors: Under LINUX, Verify is always `True`.

See also: SetVerify (45)

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`.

Errors: Under LINUX this call does nothing, because interrupts are managed by the kernel. The only allowed interrupt is 80h, the kernel entry interrupt.

See also: MSDos (42), 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.

Errors: Under LINUX, this call does nothing.

See also: `Halt` ()

MSDos

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

Description: `MSDos` executes an MS-DOS call (int 21h). This is the same as doing a `Intr` call with an interrupt number of 21h.

Errors: None.

See also: Intr (42)

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 (44), FindFirst (36), FindNext (37), UnPackTime (45)

```
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 under DOS. 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.

Errors: Under Linux, this call exists but is not implemented, i.e. it does nothing.

See also: GetCBreak (38)

SetDate

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

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

Errors: On a LINUX machine, this is not implemented (allthough a procedure exists, it just doesn't do anything. The setting of the date is a root-only privilege, and is hence not implemented.

See also: GetDate (158), SetTime (45)

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 = 01h`
- `Hidden = 02h`
- `SysFile = 04h`
- `VolumeId = 08h`
- `Directory = 10h`
- `Archive = 20h`
- `AnyFile = 3fh`

Errors: Errors are reported in `DosError`. Under LINUX the call exists, but is not implemented, i.e. it does nothing.

See also: GetFAttr (39)

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.

Errors: Errors are reported in `DosError`

See also: GetFTime (40), PackTime (43), UnPackTime (45)

SetIntVec

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

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

Errors: Under LINUX, this call exists but is not implemented, the kernel manages all interrupts.

See also: GetIntVec (41)

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.

Errors: this call exists, but is not implemented on LINUX, as setting the time is a root-only privilege.

See also: GetTime (162), SetDate (44)

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.

Errors: Under LINUX, Verify is always `True`.

See also: SetVerify (45)

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.

Errors: Under LINUX this call does nothing, as the interrupt vectors are managed by the kernel.

See also: Exec (35), SetIntVec (44)

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 (40), FindFirst (36), FindNext (37), PackTime (43)

For an example, see PackTime (43).

Chapter 3

The GETOPTS unit.

This document describes the GETOPTS unit for Free Pascal. It was written for LINUX by Michaël Van Canneyt. It also works under DOS and Tp7. 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.

3.1 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.

3.2 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 (47) 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 (47), `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 (47), `getopt` (3)

```
program testopt;

{ Program to depmonstrate the getopts 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 getopts;

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',@theopts[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 : ',theopts[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 4

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 three sections. The first section is an introduction to the GO32 unit. The second section lists the pre-defined constants, types and variables. The third section describes the functions which appear in the interface part of the GO32 unit.

4.1 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 former 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

4.2 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 relys 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 (67), get_ds (68), gett_ss (75), allocate_ldt_descriptors (60), free_ldt_descriptor (66), segment_to_descriptor (81), get_next_selector_increment_value (69), get_segment_base_address (74), set_segment_base_address (84), set_segment_limit (84), create_code_segment_alias_descriptor (63)

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 recommened 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 (65), dosmemput (66), dosmemmove (65), dosmemfillchar (63), dosmemfillword (64), mem[]-arrays, seg_move (82), seg_fillchar (80), seg_fillword (81).

I/O port access

The I/O port access is done via the various inportb (77), outportb (79) functions which are available. Additionally Free Pascal supports the Turbo Pascal PORT[]-arrays but it is by no means recommened to use them, because they're only for compatibility purposes. See also: outportb (79), inportb (77), 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 (67), get_ds (68), get_ss (75)

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 explicitely called by an 'int' instruction and are a bit comparable to normal functions. Hardware interrupts come from external devices like the keyboard or mouse. These functions 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 acomplished 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.

```
Program Keyclick;

uses crt,
     go32;

const kbdint = $9;

var oldint9_handler : tseginfo;
    newint9_handler : tseginfo;

    clickproc : pointer;

{ ASMMODE DIRECT}
procedure int9_handler; assembler;
asm
   cli
   pushal
   movw %cs:INT9_DS, %ax
   movw %ax, %ds
   movw %ax, %es
   movw U_GO32_DOSMEMSELECTOR, %ax
   movw %ax, %fs
   call *_CLICKPROC
   popal

   ljmp %cs:OLDHANDLER

INT9_DS: .word 0
OLDHANDLER:
         .long 0
         .word 0
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);
      asm
          movw %ds, %ax
          movw %ax, INT9_DS
          movl _OLDINT9_HANDLER, %eax
          movl %eax, OLDHANDLER
          movw 4+_OLDINT9_HANDLER, %ax
          movw %ax, 4+OLDHANDLER
      end;
      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 (82) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by set_rm_interrupt (84). See also set_rm_interrupt (84), get_rm_interrupt (73), set_pm_interrupt (82), get_pm_interrupt (70), lock_data (78), lock_code (78), enable (66), disable (63), outportb (79) 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 (85) and it's linear address via transfer_buffer (??). Another way is to allocate a completely new DOS memory area via the global_dos_alloc (75) function for your use and supply its real mode address. See also: tb_size (85), transfer_buffer (??). global_dos_alloc (75), global_dos_free (77), realintr (80) The following examples illustrate the use of software interrupts.

```
Program softint;

uses go32;

var r : trealregs;

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

```
Program rmpm_int;

uses crt, go32;

{ ASMMODE DIRECT}

var r : trealregs;
    axreg : Word;

    oldint21h : tseginfo;
    newint21h : tseginfo;

procedure int21h_handler; assembler;
asm
   cmpw $0x3001, %ax
   jne CallOld
   movw $0x3112, %ax
   iret

CallOld:
   ljmp %cs:OLDHANDLER

OLDHANDLER: .long 0
            .word 0
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);
     asm
        movl _OLDINT21H, %eax
        movl %eax, OLDHANDLER
        movw 4+_OLDINT21H, %ax
        movw %ax, 4+OLDHANDLER
     end;
     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', lo(axreg),'.',hi(axreg), ' detected');
     Writeln;
     Writeln('Now you can see that there''s',
             ' a distinction between the two ways of ');
     Writeln('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 conditions:

- interrupts disabled
- `%CS:%EIP` = 48 bit pointer specified in the original call to get_rm_callback (70)
- `%DS:%ESI` = 48 bit pointer to 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 (70), free_rm_callback (67), lock_code (78), lock_data (78)

4.3 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_dpmi    = 4; { DPMI (for example \dos box or 386Max) }
```

Note: GO32V2 *always* creates DPMI programs, so you need a suitable DPMI host like CWSDPMI.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 trealregs 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.

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

NOTE: The value of a field is -1 (0ffffffffh) if 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 recommened 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.

4.4 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 (69) function.

Errors: Check the `int31error` variable.

See also: free_ldt_descriptor (66), get_next_selector_increment_value (69), segment_to_descriptor (81), create_code_segment_alias_descriptor (63), set_segment_limit (84), set_segment_base_address (84)

```
Program sel_des;

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);
  clreol;
  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 is used to free the memory block afterwards or use this handle for other purposes. So this block can't be deallocated and is henceforth unusuable ! This function doesn't allocate any descriptors for this block, it's the applications resposibility to allocate and initialize for accessing this memory.

Errors: Check the `int31error` variable.

See also: free_memory_block (67)

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 (85), transfer_buffer (??), copytodos (63)

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 (85), transfer_buffer (??), copyfromdos (62)

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: selector.

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 (60), set_segment_limit (84), set_segment_base_address (84)

disable

Declaration: `Procedure disable ;`

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

Errors: None.

See also: enable (66)

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 (66), dosmemget (65), dosmemmove (65)dosmemmove, dosmemfillword (64), seg_move (82), seg_fillchar (80), seg_fillword (81)

```
Program textmess;

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 (66), dosmemget (65), dosmemmove (65), dosmemfillchar (63), seg_move (82), seg_fillchar (80), seg_fillword (81)

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: dosmemput (66), dosmemmove (65), dosmemfillchar (63), dosmemfillword (64), seg_move (82), seg_fillchar (80), seg_fillword (81)

For an example, see global_dos_alloc (75).

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: dosmemput (66), dosmemget (65), dosmemfillchar (63), dosmemfillword (64) seg_move (82), seg_fillchar (80), seg_fillword (81)

For an example, see seg_fillchar (80).

dosmemput

Declaration: `Procedure dosmemput (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 (65), dosmemmove (65), dosmemfillchar (63), dosmemfillword (64), seg_move (82), seg_fillchar (80), seg_fillword (81)

For an example, see global_dos_alloc (75).

enable

Declaration: `Procedure enable ;`

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

Errors: None.

See also: disable (63)

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 (60) 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 (60), get_next_selector_increment_value (69)

For an example, see allocate_ldt_descriptors (60).

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 (62) . 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 (62)

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 (70) function. Parameters:

intaddr: real mode address buffer returned by get_rm_callback (70) .

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

Errors: Check the `int31error` variable.

See also: set_rm_interrupt (84), get_rm_callback (70)

For an example, see get_rm_callback (70).

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 (68), get_ss (75)

For an example, see set_pm_interrupt (82).

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 (82)

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 (67), get_ss (75)

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 resposibility 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 (60), set_segment_limit (84), set_segment_base_address (84)

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 DPMI 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 DPMI host can be obtained with the get_page_size (70) function.

Errors: Check the `int31error` variable.

See also: get_page_size (70)

```
Program meminf;

uses go32;

var meminfo : tmeminfo;

begin
get_meminfo(meminfo);
if (int31error <> 0) then
 begin
```

```
  Writeln('Error getting DPMI memory information... Halting');
  Writeln('DPMI error number : ', int31error);
 end
else
 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.
```

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 (60). Parameters: None. Return value: Selector increment value. Notes: Because allocate_ldt_descriptors (60) 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 (60), free_ldt_descriptor (66)

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 (68)

For an example, see get_meminfo (68).

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 (82), set_rm_interrupt (84), get_rm_interrupt (73)

For an example, see set_pm_interrupt (82).

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 (67)

```
Program callback;

uses crt,
     go32;

const mouseint = $33;

var mouse_regs     : trealregs;
    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;

{ ASMMODE DIRECT}
procedure callback_handler; assembler;
asm
   pushw %es
   pushw %ds
   pushl %edi
   pushl %esi
   cmpl $1, _USERPROC_INSTALLED
   je .LNoCallback
   pushal
   movw %es, %ax
   movw %ax, %ds
   movw U_GO32_DOSMEMSELECTOR, %ax
   movw %ax, %fs
   call *_USERPROC_PROC
   popal
.LNoCallback:

   popl %esi
   popl %edi
   popw %ds
   popw %es

   movl (%esi), %eax
   movl %eax, %es: 42(%edi)
   addw $4, %es: 46(%edi)
   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 Mircosoft compatible mouse found');
       Write('A Mircosoft compatible mouse driver is');
       writeln(' necessary to run this example');
       halt;
    end;
    if (r.bx = $ffff) then mouse_numbuttons := 2
    else mouse_numbuttons := r.bx;
    Writeln(mouse_numbuttons,
            ' button Mircosoft 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_seginfo, sizeof(mouse_seginfo));
    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
              { write mouse state info }
              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 (84), set_pm_interrupt (82), get_pm_interrupt (70)

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 (74)

```
Program getrunmd;

uses go32;

begin
{
  depending on the detected environment,
  we simply write another message
}
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_dpmi    :
    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.

Return values: Linear base address of specified descriptor.

Errors: Check the `int31error` variable.

See also: allocate_ldt_descriptors (60), set_segment_base_address (84), allocate_ldt_descriptors (60), set_segment_limit (84), get_segment_limit (75)

For an example, see allocate_ldt_descriptors (60).

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 (60), set_segment_limit (84), set_segment_base_address (84), get_segment_base_address (74),

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 (68), get_cs (67)

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 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 an unique selector. The returned selector should only be freed by a global_dos_free (77) call.

Errors: Check the `int31error` variable.

See also: global_dos_free (77)

```
Program buffer;

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 = 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';
     dosmemput(segment, 0, infobuf, sizeof(infobuf));
     r.ax := $4f00; r.es := segment;
     realintr($10, r);
     dosmemget(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), '
     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 (75).

Errors: Check the `int31error` variable.

See also: global_dos_alloc (75)

For an example, see global_dos_alloc (75).

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 (79), inportw (77), inportl (77)

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 (79), inportb (77), inportw (77)

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 (79) inportb (77), inportl (77)

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 (78), lock_data (78), unlock_linear_region (85), unlock_data (85), unlock_code (85)

For an example, see get_rm_callback (70).

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 (78), lock_code (78), unlock_linear_region (85), unlock_data (85), unlock_code (85)

For an example, see get_rm_callback (70).

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 (78), lock_code (78), unlock_linear_region (85), unlock_data (85), unlock_code (85)

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 (77), outportl (79), outportw (79)

```
program outport;

uses crt, go32;

begin
 { turn on speaker }
 outportb($61, $ff);
 { wait a little bit }
 delay(50);
 { turn it off again }
 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 (77), outportw (79), outportb (79)

For an example, see outportb (79).

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 (77), outportl (79), outportb (79)

For an example, see outportb (79).

realintr

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

Description: Simulates an interrupt in real mode. Parameters:

intnr: 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 intnr. The real mode handler must return by executing an IRET.

Errors: Check the `int31error` variable.

See also:

```
Program flags;

uses go32;

var r : trealregs;

begin
     r.ax := $5300;
     r.bx := 0;
     realintr($15, r);
     { check if carry clear and write a suited message }
     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 (82), seg_fillword (81), dosmemfillchar (63), dosmemfillword (64), dosmemget (65), dosmemput (66), dosmemmove (65)

```
Program svgasel;

uses go32;

var vgasel : Word;
    r : trealregs;

begin
  r.eax := $13; realintr($10, r);
  vgasel := segment_to_descriptor($A000);
  { simply fill the screen memory with color 15 }
  seg_fillchar(vgasel, 0, 64000, #15);
  readln;
 { back to text mode }
  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 (82), seg_fillchar (80), dosmemfillchar (63), dosmemfillword (64), dosmemget (65), dosmemput (66), dosmemmove (65)

For an example, see allocate_ldt_descriptors (60).

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 (60) and change the base address as necessary.

Errors: Check the `int31error` variable.

See also: allocate_ldt_descriptors (60), free_ldt_descriptor (66), set_segment_base_address (84)

For an example, see seg_fillchar (80).

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 (80), seg_fillword (81), dosmemfillchar (63), dosmemfillword (64), dosmemget (65), dosmemput (66), dosmemmove (65)

For an example, see allocate_ldt_descriptors (60).

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 (67)

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 (70), set_rm_interrupt (84), get_rm_interrupt (73)

```
Program int_pm;

uses crt, go32;

const int1c = $1c;

var oldint1c : tseginfo;
    newint1c : tseginfo;
    int1c_counter : Longint;

{ ASMMODE DIRECT}
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
INT1C_DS: .word 0
end;

var i : Longint;

begin
     newint1c.offset := @int1c_handler;
     newint1c.segment := get_cs;
     get_pm_interrupt(int1c, oldint1c);
     asm
        movw %ds, %ax
        movw %ax, INT1C_DS
     end;
     Writeln('-- Press any key to exit --');
     set_pm_interrupt(int1c, newint1c);
     while (not keypressed) do begin
           gotoxy(1, wherey);
           write('Number of interrupts occured : ',
                 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 (70).

Errors: Check the `int31error` variable.

See also: get_rm_interrupt (73), set_pm_interrupt (82), get_pm_interrupt (70), get_rm_callback (70)

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 (60), get_segment_base_address (74), allocate_ldt_descriptors (60), set_segment_limit (84), get_segment_base_address (74), get_segment_limit (75)

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 (60), set_segment_base_address (84), get_segment_limit (75), set_segment_limit (84)

For an example, see allocate_ldt_descriptors (60).

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 (??), copyfromdos (62) copytodos (63)

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 (85), unlock_data (85), lock_linear_region (78), lock_data (78), lock_code (78)

For an example, see get_rm_callback (70).

unlock_data

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

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

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 (85), unlock_code (85), lock_linear_region (78), lock_data (78), lock_code (78)

For an example, see get_rm_callback (70).

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 (85), unlock_code (85), lock_linear_region (78), lock_data (78), lock_code (78)

Chapter 5

The GRAPH unit.

This document describes the GRAPH unit for Free Pascal, under DOS. The unit was first written for DOS by Florian klämpfl. This chapter is divided in three 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.

5.1 Introduction

Requirements

The unit Graph exports functions and procedures for graphical output. It requires at least a VESA compatible VGA-Card or a VGA-Card with software-driver (min. **512Kb** video memory). Before the graph unit can be used, be sure your graphics adapter supports the VESA-Standard. Otherwise in the most cases you can try to use a VESA-TSR to make your adapter VESA compatible (e.g. UNIVBE).

5.2 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;
```

5.3 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 (88), Ellipse (90) GetArcCoords (90), PieSlice (98), Sector (99)

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 (88), Rectangle (99)

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 (88), Rectangle (99)

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 (90), Arc (88) GetArcCoords (90), PieSlice (98), Sector (99)

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 (89), SetBkColor (100)

ClearViewPort

Declaration: `Procedure ClearViewPort ;`

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

Errors: None.

See also: ClearDevice (89), SetViewPort (103), SetBkColor (100)

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 (96)

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 (96)

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 (88), seepBar3D, Rectangle (99)

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.

Errors: None.

See also: Arc (88) Circle (88), FillEllipse (90)

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 (88) Circle (88), GetArcCoords (90), PieSlice (98), Sector (99)

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 (88), seepBar3D, Rectangle (99)

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 (100), SetBkColor (100)

GetArcCoords

Declaration: `Procedure GetArcCoords (Var ArcCoords : ArcCoordsType);`

Description: `GetArcCoords` returns the coordinates of the latest `Arc` or `Ellipse` call.

Errors: None.

See also: Arc (88), Ellipse (90)

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 (96), SetAspectRatio (100)

GetBkColor

Declaration: `Function GetBkColor : Word;`

Description: `GetBkColor` returns the current background color (the palette entry).

Errors: None.

See also: GetColor (91), SetBkColor (100)

GetColor

Declaration: `Function GetColor : Word;`

Description: `GetColor` returns the current drawing color (the palette entry).

Errors: None.

See also: GetColor (91), SetBkColor (100)

GetDefaultPalette

Declaration: `Procedure GetDefaultPalette (Var Palette : PaletteType);`

Description: Returns the current palette in `Palette`.

Errors: None.

See also: GetColor (91), GetBkColor (91)

GetDriverName

Declaration: `Function GetDriverName : String;`

Description: `GetDriverName` returns a string containing the name of the current driver.

Errors: None.

See also: GetModeName (93), InitGraph (96)

GetFillPattern

Declaration: `Procedure GetFillPattern (Var FillPattern : FillPatternType);`

Description: `GetFillPattern` returns an array with the current fill-pattern in `FillPattern`

Errors: None

See also: SetFillPattern (101)

GetFillSettings

Declaration: `Procedure GetFillSettings (Var FillInfo :  FillSettingsType);`

Description: `GetFillSettings` returns the current fill-settings in `FillInfo`

Errors: None.

See also: SetFillPattern (101)

GetGraphMode

Declaration: `Function GetGraphMode :  Integer;`

Description: `GetGraphMode` returns the current graphical modus

Errors: None.

See also: InitGraph (96)

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 (95), PutImage (98)

GetLineSettings

Declaration: `Procedure GetLineSettings (Var LineInfo :  LineSettingsType);`

Description: `GetLineSettings` returns the current Line settings in `LineInfo`

Errors: None.

See also: SetLineStyle (102)

GetMaxColor

Declaration: `Function GetMaxColor :  Word;`

Description: `GetMaxColor` returns the maximum color-number which can be set with `SetColor`

Errors: None.

See also: SetColor (100), GetPaletteSize (93)

GetMaxMode

Declaration: `Function GetMaxMode :  Word;`

Description: `GetMaxMode` returns the highest modus for the current driver.

Errors: None.

See also: InitGraph (96)

GetMaxX

Declaration: `Function GetMaxX : Word;`

Description: `GetMaxX` returns the maximum horizontal screen length

Errors: None.

See also: GetMaxY (93)

GetMaxY

Declaration: `Function GetMaxY : Word;`

Description: `GetMaxY` returns the maximum number of screen lines

Errors: None.

See also: GetMaxY (93)

GetModeName

Declaration: `Function GetModeName (Var modus :  Integer) :  String;`

Description: Returns a string with the name of modus `Modus`

Errors: None.

See also: GetDriverName (91), InitGraph (96)

GetModeRange

Declaration:
```
Procedure GetModeRange (Driver :  Integer;
LoModus, HiModus:  Integer);
```

Description: `GetModeRange` returns the Lowest and Highest modus of the currently installed driver.

Errors: None.

See also: InitGraph (96)

GetPalette

Declaration: `Procedure GetPalette (Var Palette :  PaletteType);`

Description: `GetPalette` returns in `Palette` the current palette.

Errors: None.

See also: GetPaletteSize (93), SetPalette (102)

GetPaletteSize

Declaration: `Function GetPaletteSize :  Word;`

Description: `GetPaletteSize` returns the maximum number of entries in the current palette.

Errors: None.

See also: GetPalette (93), SetPalette (102)

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 (103), SetTextJustify (102)

GetViewSettings

Declaration: `Procedure GetViewSettings (Var ViewPort : ViewPortType);`

Description: `GetViewSettings` returns the current view-port and clipping settings in `ViewPort`.

Errors: None.

See also: SetViewPort (103)

GetX

Declaration: `Function GetX : Integer;`

Description: `GetX` returns the X-coordinate of the current position of the graphical pointer

Errors: None.

See also: GetY (94)

GetY

Declaration: `Function GetY : Integer;`

Description: `GetY` returns the Y-coordinate of the current position of the graphical pointer

Errors: None.

See also: GetX (94)

GraphDefaults

Declaration: `Procedure GraphDefaults ;`

Description: `GraphDefaults` resets all settings for view-port, 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 (103), SetFillStyle (101), SetColor (100), SetBkColor (100), SetLineStyle (102)

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 (95)

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. Except for all operations which draw something on the screen, the following procedures also can produce a `GraphResult` different from zero:

- InstallUserFont (96)
- SetLineStyle (102)
- SetWriteMode (104)
- SetFillStyle (101)
- SetTextJustify (102)
- SetGraphMode (101)
- SetTextStyle (103)

Errors: None.

See also: GraphErrorMsg (95)

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 (92)

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 87), DetectGraph (89), CloseGraph (89), GraphResult (95)

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 (96), InstallUserFont (96)

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 (96), InstallUserDriver (96)

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 (97), LineTo (97)

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 (97), LineTo (97)

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 (97), Line (97)

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 (97)

MoveTo

Declaration: `Procedure MoveTo (X,Y : Integer;`

Description: `MoveTo` moves the pointer to the point `(X,Y)`.

Errors: None.

See also: MoveRel (97)

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 (98)

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 (98)

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 (88), Circle (88), Sector (99)

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 (95), GetImage (92)

PutPixel

Declaration: `Procedure PutPixel (X,Y : Integer; Color : Word);`

Description: Puts a point at `(X,Y)` using color `Color`

Errors: None.

See also: GetPixel (94)

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 (88), Bar3D (88)

RegisterBGIDriver

Declaration: `Function RegisterBGIDriver (Driver : Pointer) : Integer;`

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserDriver (96), RegisterBGIFont (99)

RegisterBGIFont

Declaration: `Function RegisterBGIFont (Font : Pointer) : Integer;`

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserFont (96), RegisterBGIDriver (99)

RestoreCRTMode

Declaration: `Procedure RestoreCRTMode ;`

Description: Restores the screen modus which was active before the graphical modus was started.

Errors: None.

See also: InitGraph (96)

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 (88), Circle (88), PieSlice (98)

SetActivePage

Declaration: `Procedure SetActivePage (Page :  Word);`

Description: Sets `Page` as the active page for all graphical output.

Errors: None.

See also:

SetAllPallette

Declaration: `Procedure SetAllPallette (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 (93)

SetAspectRatio

Declaration: `Procedure SetAspectRatio (Xasp,Yasp :  Word);`

Description: Sets the aspect ratio of the current screen to `Xasp/Yasp`.

Errors: None

See also: InitGraph (96), GetAspectRatio (91)

SetBkColor

Declaration: `Procedure SetBkColor (Color :  Word);`

Description: Sets the background color to `Color`.

Errors: None.

See also: GetBkColor (91), SetColor (100)

SetColor

Declaration: `Procedure SetColor (Color :  Word);`

Description: Sets the foreground color to `Color`.

Errors: None.

See also: GetColor (91), SetBkColor (100)

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 (91), SetFillStyle (101)

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 (101)

SetGraphBufSize

Declaration: `Procedure SetGraphBufSize (BufSize : Word);`

Description: `SetGraphBufSize` sets the graphical buffer size. The default size is 4Kb

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 (96)

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 (92)

SetPalette

Declaration: `Procedure SetPalette (ColorNr : Word; NewColor : ShortInt);`

Description: `SetPalette` changes the `ColorNr`-th entry in the palette to `NewColor`

Errors: None.

See also: SetAllPallette (100), SetRGBPalette (102)

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: SetAllPallette (100), SetPalette (102)

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 (98), OutTextXY (98)

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 (94)

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 (103)

SetViewPort

Declaration: `Procedure SetViewPort (X1,Y1,X2,Y2 : Integer; Clip : Boolean);`

Description: Sets the current graphical view-port (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 view-port (window) will be clipped (i.e. not drawn). Coordinates specified after this call are relative to the top-left corner of the view-port.

Errors: None.

See also: GetViewSettings (94)

SetVisualPage

Declaration: `Procedure SetVisualPage (Page : Word);`

Description: `SetVisualPage` sets the video page to page number `Page`.

Errors: None

See also: SetActivePage (100)

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 (104)

TextWidth

Declaration: `Function TextWidth (S : String) : Word;`

Description: `TextHeight` returns the width (in pixels) of the string `S` in the current font and text-size.

Errors: None.

See also: TextHeight (104)

Chapter 6

The HEAPTRC unit.

This chapter describes the HEAPTRC unit for Free Pascal. It was written by Pierre Muller.

6.1 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.

6.2 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.

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 08052C48 invalid
Wrong size : 128 allocated 64 freed
  0x0804C29C
  0x080509E2
  0x080480A4
  0x00000000
Heap dump by heaptrc unit
13 memory blocks allocated : 1416/1424
6 memory blocks freed     : 708/712
7 unfreed memory blocks : 708
True heap size : 2097152
True free heap : 2096040
Should be : 2096104
Call trace for block 0x08052C48 size 128
  0x080509D6
  0x080480A4
Call trace for block 0x08052B98 size 128
  0x08050992
  0x080480A4
Call trace for block 0x08052AE8 size 128
  0x08050992
  0x080480A4
Call trace for block 0x08052A38 size 128
  0x08050992
  0x080480A4
Call trace for block 0x08052988 size 128
  0x08050992
  0x080480A4
Call trace for block 0x080528D8 size 128
  0x08050992
  0x080480A4
Call trace for block 0x080528A0 size 4
  0x08050961
  0x080480A4
```

6.3 Constants, Types and variables

The `FillExtraInfoType` is a procedural type used in the SetExtraInfo (108) call.

```
type
    FillExtraInfoType = procedure(p : pointer);
```

The following typed constants allow to fine-tune the standard dump of the memory usage by DumpHeap (107):

```
const
  tracesize = 8;
  quicktrace : boolean = true;
  HaltOnError : boolean = true;
  keepreleased : boolean = false;
```

`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 wil 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.

6.4 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 (107)

MarkHeap

Declaration: `procedure MarkHeap;`

Description: `MarkHeap` marks all memory blocks with a special signature. You can use this if you think that you corruped the memory.

Errors: None.

See also: DumpHeap (107)

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 (107) is executed.

See also: DumpHeap (107)

```
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.
```

Chapter 7

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.

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.

7.1 Types, Constants and variables :

Variables

```
Var
  IPCerror : longint;
```

The `IPCerror` variable is used to report errors, by all calls.

Constants

```
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 (119) 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 (126) call.

Types

```
Type
   TKey   = Longint;
```

TKey is the type returned by the ftok (115) 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 strucure is used in the shmctl (126) call to set or retrieve settings concerning shared memory.

```
type
  PSHMinfo = ^TSHMinfo;
  TSHMinfo = record
    shmmax : longint;
    shmmin : 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 programer 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 (116) 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 (120) call, and contains all data concerning a semahore.

```
Type
  PSEMbuf = ^TSEMbuf;
  TSEMbuf = record
    sem_num : word;
    sem_op  : integer;
    sem_flg : integer;
  end;
```

The TSEMbuf record us use in the semop (119) 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 diirectly.

```
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 (120) call.

7.2 Functions and procedures

ftok

Declaration: `Function ftok (Path : String; ID : char) : TKey;`

Description: `ftok` returns a key that can be used in a semget (119), shmget (125) or msgget (115) 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 couterpart, 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 (119), shmget (125), msgget (115)

For an example, see msgctl (116), semctl (120), shmctl (126).

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_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If 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 `OR`ed 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 (115), msgsnd (115), msgrcv (116), msgctl (116), `semget` (2)

For an example, see msgctl (116).

msgsnd

Declaration: `Function msgsnd(msqid:longint; msgp: PMSGBuf; msgsz: longint; msgflg:longint): Boolean;`

Description: `msgsend` 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. `msgsiz` 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_NOWAIT If 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 (115), msgrcv (116), seefmsgctl

For an example, see msgctl (116).

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_NOERROR If 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 (115), msgsnd (115), msgctl (116)

For an example, see msgctl (116).

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_STAT In 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_RMID If 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 (115), msgsnd (115), msgrcv (116)

```
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 : ',ipcerror);
  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(elete)');
  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.
```

semget

Declaration: `Function semget(key:Tkey; nsems:longint; semflg:longint): longint;`

Description: `msgget` 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_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If 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 `OR`ed 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 (115), semop (119), semctl (120)

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_num The number of the semaphore in the set on which the operation must be performed.

sem_op The 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 substracted 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_flg Optional 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 (119), semctl (120)

semctl

Declaration: `Function semctl(semid:longint; semnum:longint; cmd:longint; var arg: tsemun): longint;`

Description: `semctl` performs various operations on the semaphore `semnum` w ith 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_STAT In this case, the arg record should have it's `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_SET In this case, the `arg` record should have it's `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_RMID If this is specified, the semaphore set is removed from 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 bumber of processes waiting for resources.

GETPID`semctl` returns the process ID of the process that performed the last semop (119) 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 (119), semop (119)

```
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 : TSEMid_ds;

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 I,cnt : longint;

begin
  cnt:=getmembercount(ID);
  Writeln ('Semaphore ',ID,' has ',cnt,' Members');
  For I:=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(elete)');
  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_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If 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 `OR`ed 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 costants:

SHM_RND The 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 (125), shmdt (126), shmctl (126)

For an example, see shmctl (126).

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 (125).

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 (125), shmat (125), shmctl (126)

shmctl

Declaration: `Function shmctl(shmid:longint; cmd:longint; buf: pshmid_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_SETapplies the values in the `ipc_perm` record that `buf` points to, to the shared memory block.

IPC_RMIDthe 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 (125), shmat (125), shmdt (126)

```
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
  strpcopy (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 (shmid,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 (shmid,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');
  shmid := shmget(key,segsize,IPC_CREAT or IPC_EXCL or 438);
  If shmid=-1 then
    begin
    Writeln ('Shared memory exists. Opening as client');
    shmid := shmget(key,segsize,0);
    If shmid = -1 then
      begin
      Writeln ('shmget : Error !',ipcerror);
      halt(1);
      end
    end
  else
    Writeln ('Creating new shared memory segment.');
  segptr:=shmat(shmid,nil,0);
  if longint(segptr)=-1 then
    begin
    Writeln ('Shmat : error !',ipcerror);
    halt(1);
    end;
  case upcase(paramstr(1)[1]) of
    'W' : writeshm (shmid,segptr,paramstr(2));
    'R' : readshm (shmid,segptr);
    'D' : removeshm(shmid);
    'M' : changemode (shmid,paramstr(2));
  else
    begin
    writeln (paramstr(1));
    usage;
    end;
  end;
end.
```

Chapter 8

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 2 sections:

- The first section lists all constants, types and variables, as listed in the interface section of the LINUX unit.
- The second section describes all procedures and functions in the LINUX unit.

8.1 Type, Variable and Constant declarations

Types

PGlob and TGlob are 2 types used in the Glob (163) function:

```
PGlob = ^TGlob;
TGlob = record
  Name : PChar;
  Next : PGlob;
  end;
```

The following types are used in the signal-processing procedures.

```
{$Packrecords 1}
SignalHandler   = Procedure ( Sig : Integer);cdecl;
PSignalHandler  = SignalHandler;
SignalRestorer  = Procedure;cdecl;
PSignalrestorer = SignalRestorer;
SigActionRec = Record
  Sa_Handler  : Signalhandler;
  Sa_Mask     : Longint;
  Sa_flags    : Integer;
  Sa_Restorer : SignalRestorer;
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;
     blksze : 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 (172) and ReadDir (174) 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 (174) 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 iotcl() 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 (185) call to set access and modificaton time of a file.

```
utimbuf = record
  actime,modtime : Longint;
  end;
```

For the Select (175) call, the following 4 types are needed:

```
FDSet = Array [0..31] of longint;
PFDSet = ^FDSet;
TimeVal = Record
   sec,usec : Longint;
end;
PTimeVal = ^TimeVal;
```

The Uname (184) 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 ;
  TIOCGLCKTRMIOS = $5456 ;
  TIOCSLCKTRMIOS = $5457 ;
  TIOCSERGSTRUCT = $5458  ;
  TIOCSERGETLSR   = $5459  ;
  TIOCSERGETMULTI = $545A  ;
  TIOCSERSETMULTI = $545B  ;
  TIOCMIWAIT = $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 ca 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 (154) 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_minux_V2 = $2468;
fs_msdos    = $4d44;
fs_nfs      = $6969;
fs_proc     = $9fa0;
fs_xia      = $012FD16D;
```

the FLock (153) call uses the following mode constants :

```
LOCK_SH = 1;
LOCK_EX = 2;
LOCK_UN = 8;
LOCK_NB = 4;
```

8.2 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_OK User has read rights.

W_OK User has write rights.

X_OK User has execute rights.

F_OK User 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_eaccess The requested access is denied, either to the file or one of the directories in its path.

sys_einval`Mode` 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_enomem Insufficient kernel memory.

sys_eloop`Path` has a circular symbolic link.

See also: Chown (139), Chmod (140), `Access` (2)

```
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.
```

AssignPipe

Declaration: `Procedure AssignPipe (Pipe_in, Pipe_out : Text);`

Description: `AssignePipe` 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`. Reading and writing happens through the usual `Readln(Pipe_in,...)` and `Writeln (Pipe_out,...)` procedures.

Errors: `LinuxError` is used to report errors:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: POpen (173), MkFifo (171), `pipe` (2)

```
Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses linux;
```

```
Var pipi,pipo : Text;
    s : String;

begin
  Writeln ('Assigning Pipes.');
  assignpipe(pipi,pipo);
  if linuxerror<>0 then
    Writeln('Error assigning pipes !');
  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: `Procedure AssignStream (StreamIn,StreamOut : Text; Const prog : String);`

Description: `AssignStream` creates a 2 pipes, i.e. two file objects, one for input, one for output, the other ends of these pipes are connected to standard input and and output 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 be read from `StreamIn`. Reading and writing happens through the usual `Readln(StreamIn,...)` and `Writeln (StreamOut,...)` procedures.

Errors: `LinuxError` is used to report errors:

sys_emfile 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 (136), POpen (173),`pipe` (2)

```
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;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 (142), FExpand (153), `Basename` (1)

```
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 (139), CFSetISpeed (139), `termios` (2)

For an example, see TCGetAttr (182).

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 (139), CFMakeRaw (139), `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 (139), CFMakeRaw (139), `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_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_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_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: Chmod (140), Access (135), `Chown` (() 2)

```
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_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_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 (139), Access (135), `Chmod` (() 2)

```
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.
```

CloseDir

Declaration: `Function CloseDir (p:pdir) : integer;`

Description: `CloseDir` closes the directory pointed to by `p`. It returns zero if the directory was closed succesfully, -1 otherwise.

Errors: Errors are returned in LinuxError.

See also: OpenDir (172), ReadDir (174), SeekDir (174), TellDir (184), `closedir` (3)

For an example, see OpenDir (172).

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 (138), FExpand (153), `Dirname` (1)

```
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: `Procedure Dup (Var OldFile, NewFile : Text);`

Description: Makes `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile`. 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.

Errors: `Linuxerror` is used to report errors.

sys_ebadf`OldFile` hasn't been assigned.

sys_emfile Maximum number of open files for the process is reached.

See also: Dup2 (143), `Dup` (2)

```
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.');
  writeln (f,'This is written to the dup file, and flushed');flush(f);
  writeln
end.
```

Dup2

Declaration: `Procedure Dup2 (Var OldFile, NewFile : Text);`

Description: Makes `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile`. `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.

Errors: `Linuxerror` is used to report errors.

sys_ebadf`OldFile` hasn't been assigned.

sys_emfile Maximum number of open files for the process is reached.

See also: Dup (142), `Dup2` (2)

```
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.

Errors: None

See also: GetEpochTime (160), LocalToEpoch (170), GetTime (162), GetDate (158)

```
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_eacces File is not a regular file, or has no execute permission. A compononent of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: Execve (147), Execv (146), Execvp (148), Execle (145), Execlp (146), Fork (157), `execvp` (3)

```
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.
```

Execele

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_eacces File is not a regular file, or has no execute permission. A compononent of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: Execve (147), Execv (146), Execvp (148), Execl (144), Execlp (146), Fork (157), `execvp` (3)

```
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.}
  Execle ('/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_eacces File is not a regular file, or has no execute permission. A compononent of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: Execve (147), Execv (146), Execvp (148), Execle (145), Execl (144), Fork (157), `execvp` (3)

```
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_eacces File is not a regular file, or has no execute permission. A compononent of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: Execve (147), Execvp (148), Execle (145), Execl (144), Execlp (146), Fork (157), `execv` (3)

```
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 : 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`:

eaccesFile is not a regular file, or has no execute permission. A compononent of the path has no search permission.

sys_ eperm The file system is mounted *noexec*.

sys_ e2big Argument list too big.

sys_ enoexec The magic number in the file is incorrect.

sys_ enoent The file does not exist.

sys_ enomem Not enough memory for kernel.

sys_ enotdirA component of the path is not a directory.

sys_ eloop The path contains a circular reference (via symlinks).

See also: Execve (147), Execv (146), Execvp (148) Execle (145), Execl (144), Execlp (146), Fork (157), `execve` (2)

```
Program Example7;

{ Program to demonstrate the Execve 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 }
  { Envp is defined in system.inc }
  ExecVe ('/bin/ls',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_eacces File is not a regular file, or has no execute permission. A compononent of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: Execve (147), Execv (146), Execle (145), Execl (144), Execlp (146), Fork (157), `execvp` (3)

```
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.}
  { Envp is defined in the system unit. }
  Execvp ('ls',pp,envp);
end.
```

FD_Clear

Declaration: `Procedure FD_Clear (var fds:fdSet);`

Description: `FD_Clear` clears all the filedescriptors in the file descriptor set `fds`.

Errors: None.

See also: Select (175), SelectText (176), GetFS (160), FD_Clr (149), FD_Set (150), FD_IsSet (149)

For an example, see Select (175).

FD_Clr

Declaration: `Procedure FD_Clr (fd:longint;var fds:fdSet);`

Description: `FD_Clr` clears file descriptor `fd` in filedescriptor s et `fds`.

Errors: None.

See also: Select (175), SelectText (176), GetFS (160), FD_Clear (149), FD_Set (150), FD_IsSet (149)

For an example, see Select (175).

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 (175), SelectText (176), GetFS (160), FD_Clear (149), FD_Clr (149), FD_Set (150)

For an example, see Select (175).

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 (175), SelectText (176), GetFS (160), FD_Clear (149), FD_Clr (149), FD_IsSet (149)

For an example, see Select (175).

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 (150), fdRead (151), fdWrite (153), fdTruncate (152), fdFlush (150), seefF-dSeek

For an example, see fdOpen (150).

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 (150), fdClose (150), fdRead (151), fdWrite (153), fdTruncate (152), fdSeek (152)

For an example, see fdRead (151).

fdOpen

Declaration: `Function fdOpen (Var PathName;flags:longint[; Mode: longint]) : longint;`

Description: `fdOpen` opens a file in `pathname` with flags `flags` a ORed combination of `Open_Accmode`, `Open_RdOnly`, `Open_WrOnly`, `Open_RdWr`, `Open_Creat`, `Open_Excl`, `Open_NoCtty`, `Open_Trunc`, `Open_Append`, `Open_NonBlock`, `Open_NDelay`, `Open_Sync` `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 filedescriptor, or a negative value if there was an error.

Errors: Errors are returned in LinuxError

See also: fdClose (150), fdRead (151), fdWrite (153), fdTruncate (152), fdFlush (150), fdSeek (152)

```
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 (150), fdClose (150), fdWrite (153), fdTruncate (152), fdFlush (150), fdSeek (152)

```
Program Example20;

{ Program to demonstrate the fdRead and fdTruncate functions. }

Uses linux;

Const Data : string[10] = '12345687890';

Var FD : Longint;
    l : longint;

begin
  FD:=fdOpen('test.dat',open_wronly or open_creat,octal(666));
  if fd>0 then
    begin
    { Fill file with data }
    for l:=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 l:=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 of an error occurred.

Errors: Errors are returned in LinuxError.

See also: fdOpen (150), fdWrite (153), fdClose (150), fdRead (151), fdTruncate (152), fdFlush (150)

For an example, see fdOpen (150).

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 (150), fdClose (150), fdRead (151), fdWrite (153), fdFlush (150), fdSeek (152)

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 (150), fdClose (150), fdRead (151), fdTruncate (152), fdSeek (152), fdFlush (150)

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 (138), DirName (142)

```
Program Example45;

{ Program to demonstrate the FExpand function. }

Uses linux;

begin
  Writeln ('This program is in : ',FExpand(Paramstr(0)));
end.
```

FLock

Declaration: `Procedure FLock (Var F; Mode : longint);`

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.

Errors: Errors are reported in `LinuxError`.

See also: Fcntl (156), `flock` (2)

FSStat

Declaration: `Function FSStat (Path : Pathstr; Var Info : statfs) : Boolean;`

Description: Return in `Info` information about the filesystem on which the file `Path` 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_einval Invalid character in `Path`.

sys_enoent`Path` does not exist.

sys_eaccess Search permission is denied for component in `Path`.

sys_eloopA circular symbolic link was encountered in `Path`.

sys_eio An error occurred while reading from the filesystem.

See also: FStat (155), LStat (168), `statfs` (2)

```
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.
```

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 (138), DirName (142), FExpand (153)

```
Program Example46;

{ Program to demonstrate the FSearch function. }

Uses linux,strings;

begin
  Writeln ('ls is in : ',FSearch ('ls',strpas(Getenv('PATH'))));
end.
```

FStat

Declaration: `Function FStat (Path : Pathstr; Var Info : stat) : Boolean;`

Description: `FStat` gets information about the file specified in `Path`, 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: FSStat (154), LStat (168), `stat` (2)

```
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.blksze);
  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 : text, Cmd : Integer) : Integer;`

Description: Read a file's attributes. `Fd` is an assigned file. `Cmd` speciefies what to do, and is one of the following:

F_GetFd Read the close_on_exec flag. If the low-order bit is 0, then the file will remain open across execve calls.

F_GetFl Read the descriptor's flags.

F_GetOwn Get 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 (156), `Fcntl` (2)

Fcntl

Declaration: `Procedure Fcntl (Fd : text, Cmd : Integer; Arg : longint);`

Description: Read or Set a file's attributes. `Fd` is an assigned file. `Cmd` speciefies what to do, and is one of the following:

F_SetFd Set the close_on_exec flag of `Fd`. (only the least siginificant bit is used).

F_GetLk Return 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_SetLk Set 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 (156), `Fcntl` (2)

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 (162)).

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 (147), `fork` (2)

```
Program Example14;

{ Program to demonstrate the Fork and WaitPidfunction. }

Uses linux;

Var PID, ExitStatus : Longint;

begin
  Writeln ('Spawning a child');
  PID:=Fork;
  If PID=0 then
    begin
    Writeln ('Hello From the Child !!');
    Writeln ('Exiting with exit status 1 !');
    Halt (1);
    end
  Else
    begin
    Writeln ('Spawned child with PID : ',PID);
    WaitPid (PID,@ExitStatus,0);
    Writeln ('Child exited with status : ',ExitStatus shr 8);
    end;
end.
```

GetDate

Declaration: `Procedure GetDate (Var Year, Month, Day : Word) ;`

Description: Returns the current day.

Errors: None

See also: GetEpochTime (160), GetTime (162), EpochToLocal (144)

```
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.
```

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 (161),seemGetdomainname2

```
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 (161), `getegid` (2)

```
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 (159), `geteuid` (2)

```
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)

```
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 (144), GetTime (162), `time` (2)

```
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' need this file selector. Only for some calls it is needed, such as the Select (175) call or so.

Errors: In case the file was not opened, then -1 is returned.

See also: Select (175)

```
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 (158), `getgid` (2)

```
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 (158),seemGethostname2

```
Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses linux;

begin
  Writeln ('Name of this machine is : ',GetHostName);
end.
```

GetPid

Declaration: `Function GetPid : Longint;`

Description: Get the Process ID of the currently running process.

Errors: None.

See also: GetPPid (162), `getpid` (2)

```
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 (161), `getppid` (2)

```
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_esrch No process found using `which` and `who`.

sys_einval`Which` was not one of `Prio_Process, Prio_Grp or Prio_User`.

See also: SetPriority (176), Nice (171), `Getpriority` (2)

For an example, see Nice (171).

GetTime

Declaration: `Procedure GetTime (Var Hour,Minute, Second : Word) ;`

Description: Returns the current time of the day.

Errors: None

See also: GetEpochTime (160), GetDate (158), EpochToLocal (144)

```
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.
```

GetUid

Declaration: `Function GetUid : Longint;`

Description: Get the real user ID of the currently running process.

Errors: None.

See also: GetEUid (159), `getuid` (2)

```
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_enomem No memory on heap for glob structure.

othersAs returned by the opendir call, and sys_readdir.

See also: GlobFree (164), `Glob` (3)

```
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 (163)

For an example, see Glob (163).

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)

```
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 : Cadinal; Value : Longint) : boolean;`

Description: `IOperm` sets permissions on `Num` ports starting with port `From` to `Value`. The function returns `True` if the call was successfull, `False` otherwise. *Remark:*

- This works ONLY as root.
- Only the first `0x03ff` ports can be set.
- When doing a Fork (157), the permissions are reset. When doing a Execve (147) 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 (164), TTYName (184)

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 (155), S_ISLNK (166), S_ISREG (167), S_ISDIR (166), S_ISCHR (166), S_ISFIFO (166), S_ISSOCK (167)

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 (155), S_ISLNK (166), S_ISREG (167), S_ISDIR (166), S_ISBLK (165), S_ISFIFO (166), S_ISSOCK (167)

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 (155), S_ISLNK (166), S_ISREG (167), S_ISCHR (166), S_ISBLK (165), S_ISFIFO (166), S_ISSOCK (167)

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 (155), S_ISLNK (166), S_ISREG (167), S_ISDIR (166), S_ISCHR (166), S_ISBLK (165), S_ISSOCK (167)

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 (155), S_ISREG (167), S_ISDIR (166), S_ISCHR (166), S_ISBLK (165), S_ISFIFO (166), S_ISSOCK (167)

```
Program Example53;

{ Program to demonstrate the S_ISLNK function. }

Uses linux;

Var Info : Stat;

begin
  if FStat (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.
```

See also:

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 (155), S_ISLNK (166), S_ISDIR (166), S_ISCHR (166), S_ISBLK (165), S_ISFIFO (166), S_ISSOCK (167)

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 (155), S_ISLNK (166), S_ISREG (167), S_ISDIR (166), S_ISCHR (166), S_ISBLK (165), S_ISFIFO (166)

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_einval An invalid signal is sent.

sys_esrch The `Pid` or process group don't exist.

sys_eperm The effective userid of the current process doesn't math the one of process `Pid`.

See also: SigAction (177), Signal (179), `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_enoent`Path` does not exist.

See also: FStat (155), FSStat (154), `stat` (2)

```
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.blksze);
  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.blksze);
  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_exdev`OldPath` and `NewPath` are not on the same filesystem.

sys_eperm The 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_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_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only filesystem.

sys_eexist`NewPath` already exists.

sys_emlink`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 anothe entry.

sys_eperm`OldPath` points to . or .. of a directory.

See also: SymLink (180), UnLink (185), `Link` (2)

```
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 (160), EpochToLocal (144), GetTime (162), GetDate (158)

```
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 named a named pipe in the filesystem, with name `PathName` and mode Mode.

Errors: `LinuxError` is used to report errors:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: POpen (173), MkFifo (171), `mkfifo` (4)

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 (162), SetPriority (176), `Nice` (2)

```
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.
```

OpenDir

Declaration: `Function OpenDir (f:pchar) : 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 (142), ReadDir (174), SeekDir (174), TellDir (184), `opendir` (3)

```
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.
```

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 (173)

For an example, see POpen (173)

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 stdinput. 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 (173). 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 (136), popen (3) , PClose (173)

```
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');
  rewrite (f);
  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:pdir) :  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 (142), OpenDir (172), SeekDir (174), TellDir (184), `readdir` (3)

For an example, see OpenDir (172).

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 (142), ReadDir (174), OpenDir (172), TellDir (184), `seekdir` (3)

For an example, see OpenDir (172).

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 cheked to see if an exception occorred on them. You can use the functions FD_Clear (149), FD_Clr (149), FD_Set (150), FD_IsSet (149) 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 timout was reached, and no decsriptor 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 descriptot 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 (176), GetFS (160), FD_Clear (149), FD_Clr (149), FD_Set (150), FD_IsSet (149)

```
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 (175) 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 (175). `SYS_EBADF` can also mean that the file wasn't opened.

See also: Select (175), GetFS (160)

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_esrch No process found using `which` and `who`.

sys_einval`Which` 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_eaccesA non-superuser tried to a priority increase.

See also: GetPriority (162), Nice (171), `Setpriority` (2)

For an example, see Nice (171).

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 (157) or Execve (147) calls.

Errors: Errors are reported in LinuxError.

See also: POpen (173), Fork (157), Execve (147), `system` (3)

```
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 wwith status : ',S);
end.
```

SigAction

Declaration: `Procedure SigAction (Signum : 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. `SigNum` specifies the signal, and can be any signal except **SIGKILL** or **SIGSTOP**. If `Act` is non-nil, then the new action for signal `SigNum` 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 ignord during the execution of the signal handler. `Sa_Flags` Speciefies a series of flags which modify the behaviour of the signal handler. You can 'or' none or more of the following :

SA_NOCLDSTOP If signum 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_RESTART For 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_efault`Act,OldAct` point outside this process address space

sys_eintr System call was interrupted.

See also: SigProcMask (178), SigPending (178), SigSuspend (179), Kill (167), `Sigaction` (2)

```
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^.Sa_Handler:=@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 ('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 (177), SigProcMask (178), SigSuspend (179), Signal (179), Kill (167), `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_BLOCK The set of blocked signals is the union of the current set and the `SSet` argument.

SIG_UNBLOCK The signals in `SSet` are removed from the set of currently blocked signals.

SIG_SETMASK The 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_efault`SSet` or `OldSSet` point to an adress outside the range of the process.

sys_eintr System call was interrupted.

See also: SigAction (177), SigPending (178), SigSuspend (179), Kill (167), `Sigprocmask` (2)

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 (177), SigProcMask (178), SigPending (178), Signal (179), Kill (167), `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_ERR An error occurred.

See also: SigAction (177), Kill (167), `Signal` (2)

```
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.
```

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_eperm The 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_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_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only filesystem.

sys_eexist`NewPath` already exists.

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 anothe entry.

See also: Link (169), UnLink (185), `Symlink` (2)

```
Program Example22;

{ Program to demonstrate the SymLink 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 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
    Writeln ('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
    Writeln ('This shouldn''t happen');
 { Now remove new.txt also }
 If not Unlink ('new.txt') then
   Writeln ('Error when unlinking !');
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 (183), `termios` (2)

```
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 (182), `termios` (2)

For an example, see TCGetAttr (182).

TCSetPGrp

Declaration: `Function TCSetPGrp (Fd,Id:longint) : boolean;`

Description: `TCSetPGrp` 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 (183), `termios` (2)

For an example, see TCGetPGrp (183).

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 (165), IOCtl (164)

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 (142), ReadDir (174), SeekDir (174), OpenDir (172), `telldir` (3)

For an example, see OpenDir (172).

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 (140), `Umask` (2)

```
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 (161), GetDomainName (158), `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 succesfull, `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_enoentA component of the path doesn't exist.

sys_enotdirA 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 (169), SymLink (180), `Unlink` (2)

For an example, see Link (169).

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_enoentA 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 (160), Chown (139), Access (135), `utime` (() 2)

```
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 : Integer) : 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:

WNOHANGCauses `Waitpid` to return immediately if no child has exited.

WUNTRACEDCauses `WaitPid` to return also for children which are stopped, but whose status has not yet been reported.

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 (157), Execve (147), `waitpid` (2)

for an example, see Fork (157).

Chapter 9

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.

9.1 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;
```

9.2 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 10

The Mouse unit

The mouse unit provides basic Mouse handling under Dos (Go32v1 and Go32v2) Some general remarks about the mouse unit:

- 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 mouse 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.

10.1 Constants, types and variables

The following constants are defined (to be used in e.g. the GetLastButtonPress (190) 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.

10.2 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. For two-button mice, checking the status of the middle button seems to give and clear the stats of the right button.

Errors: None.

See also: GetLastButtonRelease (191)

```
{example for GetLastButtonPress and GetLastButtonRelease}

Uses Mouse, 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 parame
      GotoXY(1,22);
      Write('Since the last call to GetLastButtonRelease with this button as para
      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);
                  Writeln('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);
                  Writeln('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. For two-button mice, checking the stats of the middle button seems to give and clear the stats of the right button.

Errors: None.

See also: GetLastButtonPress (190)

For an example, see GetLastButtonPress (190).

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 (194), MPressed (194), RPressed (194), SetMousePos (196)

```
{example for GetMouseState, IsLPressed, IsRPressed and IsMPressed}

Uses Mouse, 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
          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 again visible.

Errors: None.

See also: ShowMouse (199), SetMouseHideWindow (195)

For an example, see ShowMouse (199).

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.

```
Program Mouse1;

{example for InitMouse and MouseFound}

Uses Mouse;

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 (192), MPressed (194), RPressed (194)

For an example, see GetMouseState (192).

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 (192), LPressed (194), RPressed (194)

For an example, see GetMouseState (192).

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 (192), LPressed (194), MPressed (194)

For an example, see GetMouseState (192).

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 (196), 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 (196)

```
{example for SetMouseAscii}

{warning: no error checking is performed on the input}

Uses Mouse, 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 valu
      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 (199), 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.

Errors: None.

See also: ShowMouse (199), HideMouse (193)

nputlistingmouseex/mouse9.pp

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 (192)

```
{example for SetMousePos}

Uses Mouse, 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 (195)

```
{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 Mouse, 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
      Writeln;
      Writeln('ASCII value of mouse cursor:');
      Writeln('Forground 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 of 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:

```
Uses Mouse, 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 numbe
      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: `SetMousWindow` defines a rectangle on screen with top-left corner at (`xmin,ymin`) and botto-right corner at (`xmax,ymax`), out of which the mouse cursor can't move. This procedure is simply a wrapper for the SetMouseXRange (198) and SetMouseYRange (199) procedures. While `xmin, ymin, xmax` and `ymax` are Longint parameters, only the lower 16 bits are used.

Errors: None.

See also: SetMouseXRange (198), SetMouseYRange (199)

For an example, see SetMouseXRange (198).

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 (199), SetMouseWindow (198)

```
{example for SetMouseXRange, SetMouseYRange and SetMouseWindow}

Uses Mouse, Crt;

Begin
  If MouseFound Then
    Begin
      SetMouseXRange(20*8,50*8);  {charracter 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)}

      Writeln('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 (198), SetMouseWindow (198)

For an example, see SetMouseXRange (198).

ShowMouse

Declaration: `Procedure ShowMouse ;`

Description: `ShowMouse` makes the mouse cursor visible. At the start of your progam, the mouse is invisible.

Errors: None.

See also: HideMouse (193), SetMouseHideWindow (195)

```
{example for ShowMouse and HideMouse}

Uses Mouse;

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 11

The Objects unit.

This chapter documents the objects unit. The unit was implemented by many people, and was mainly taken from the FreeVision sources.

The methods and fields that are in a `Private` part of an object declaration have been left out of this documentation.

11.1 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 }
```

11.2 Types

The follwing 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 internaly 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 11.4):

```
TYPE
   PPoint = ^TPoint;
   TPoint = OBJECT
      X, Y: Sw_Integer;
   END;
```

11.3 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 baed on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr (204)

```
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 (203)

For an example, see NewStr (203).

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 11.10.
2. `TStringCollection`, see section 11.12.
3. `TStrCollection`, see section 11.13.

Errors: None.

See also: RegisterType (204)

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 it's 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(TypeOf(MyType)^);
```

Load : Pointeris 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 it's sole argument a `PStream` type variable.

Store: Pointeris a pointer to a method that stores an instance of the object to a stream. This method should accept as it's 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;
```

```
Function 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 (206)

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 (206)

11.4 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 (208), TRect.Contains (208)

```
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 (207), Contains (208)

For an example, see TRect.Empty (207)

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 (209), Equals (208)

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 (211)

```
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 (209)

```
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 (209)

```
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 (211)

```
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 (210).

```
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 (208)

For an example, see TRect.Copy (208).

11.5 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 (212), Done (212)

For an example, see Free (212)

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 (212), Done (212)

```
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 (212) method.

The destructore Done does not free the memory occupied by the object.

Errors: None.

See also: Free (212), Init (212)

```
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.
```

11.6 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 descendrnt 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 (218)

```
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 (218), ReadStr (216)

```
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 (219), GetSize (216)

```
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 (219), GetPos (215)

```
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 (214)

```
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 (217), Reset (217)

For an example, see TDosStream.Open (224).

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 (217), Reset (217)

for an example, see TDosStream.Open (224).

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 (217), Close (217)

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 (218)

for an example, see TBufStream.Flush (226).

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 (219)

For an example, see TDosStream.Truncate (222).

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 (204).

After the object has been written, it can be read again with Get (214).

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 (214)

For an example, see TStream.Get (214);

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 (218), StrRead (214), ReadStr (216)

For an example, see TStream.StrRead (214).

TStream.WriteStr

Declaration: `Procedure TStream.WriteStr (P: PString);`

Description: `StrWrite` writes the pascal string pointed to by `P` to the stream.

Errors: None.

See also: StrWrite (218), StrRead (214), ReadStr (216)

For an example, see TStream.ReadStr (216).

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. seeek(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 (215), GetSize (216)

For an example, see TDosStream.Seek (223).

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 (220), ReadStr (216), StrRead (214)

```
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 (219), WriteStr (218), StrWrite (218)

For an example, see TStream.Read (219).

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 (219) method to read the data, and the Write (220) method to write in the current stream.

Errors: None.

See also: Read (219), Write (220)

```
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.
```

11.7 TDosStream

`TDosStream` is a stream that stores it's contents in a file. it overrides a couple of methods of `TSteam` for this.

In addition to the fields inherited from `TStream` (see section 11.6), 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 11.8.

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.

stOpenRead and write access.

Errors: On error, `Status` is set to `stInitError`, and `ErrorInfo` is set to the DOS error code.

See also: Done (222)

For an example, see TDosStream.Truncate (222).

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 (221), Close (222)

for an example, see e.g. TDosStream.Truncate (222).

TDosStream.Close

Declaration: `Pocedure TDosStream.Close; Virtual;`

Description: `Close` closes the file if it was open, and sets `Handle` to -1. Contrary to Done (222) it does not clean up the instance of `TDosStream`

Errors: None.

See also: TStream.Close (217), Init (221), Done (222)

For an example, see TDosStream.Open (224).

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 (218), GetSize (216)

```
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 (219), GetPos (215)

```
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 (222) 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 (217), Close (222)

```
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 (219), Write (225)

For an example, see TStream.Read (219).

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 (220), Read (224)

For an example, see TStream.Read (219).

11.8 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.

stOpenRead 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 (221), Done (226)

For an example see TBufStream.Flush (226).

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 (222), Init (225), Close (226)

For an example see TBufStream.Flush (226).

TBufStream.Close

Declaration: `Pocedure TBufStream.Close; Virtual;`

Description: `Close` flushes and closes the file if it was open, and sets `Handle` to -1. Contrary to Done (226) it does not clean up the instance of `TBufStream`

Errors: None.

See also: TStream.Close (217), Init (225), Done (226)

For an example see TBufStream.Flush (226).

TBufStream.Flush

Declaration: `Pocedure 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 (228) for more info on the errors.

See also: TStream.Close (217), Init (225), Done (226)

```
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 (226) or TDosStream.Truncate (222).

See also: TStream.Truncate (218), TDosStream.Truncate (222), GetSize (216)

For an example, see TDosStream.Truncate (222).

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 (219), GetPos (215)

For an example, see TStream.Seek (219);

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 (226) 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 (217), Close (226)

For an example, see TDosStream.Open (224).

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 contiunuing 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 (219), Write (228)

For an example, see TStream.Read (219).

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 (220), Read (228)

For an example, see TStream.Read (219).

11.9 TMemoryStream

The TMemoryStream object implements a stream that stores it's data in memory. The data is stored on the heap, with the possibility to specify the maximum amout of data, and the 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;                                                 Virtual;
      PROCEDURE Truncate;                                              Virtual;
      PROCEDURE Read (Var Buf; Count: Sw_Word);                        Virtual;
      PROCEDURE Write (Var Buf; Count: Sw_Word);                       Virtual;
   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 (229)

For an example, see e.g TStream.CopyFrom (220).

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 (229)

For an example, see e.g TStream.CopyFrom (220).

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 (218)

```
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 (231)

For an example, see TStream.Read (219).

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 constuctor call Init (229)) as needed.

Errors: If the stream cannot allocate more memory, then the status is set to `stWriteError`

See also: TStream.Write (220), Read (230)

For an example, see TStream.Read (219).

11.10 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 allocatiod in case `ALimit` is reached, and another element is added to the collection.

Errors: None.

See also: Load (232), Done (233)

For an example, see TCollection.ForEach (242).

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 (235).

See also: Init (232), GetItem (235), Done (233).

```
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 (232), FreeAll (237)

For an example, see TCollection.ForEach (242).

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 (240)

```
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 (214).

See also: TStream.Get (214), seeplStoreTCollection.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 (236)

```
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 (235)

```
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 (243)

```
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 occumpied by the collection itself, but it does set `Count` to zero.

Errors:

See also: DeleteAll (238), FreeItem (241)

```
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 (237), `DeletAll` doesn't call the destructor of the objects.

Errors: None.

See also: FreeAll (237), Delete (240)

```
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 (241),

```
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 (244), AtPut (244),

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 (239) call is provided.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: AtDelete (242), Free (239)

```
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 (243) is called with `CoIndexError`.

See also: Free (239), AtDelete (242)

```
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 (241), 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 (243) is called with `CoIndexError`.

See also: Delete (240)

```
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 (236), LastThat (235)

```
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 (232)

For an example, see Pack (236).

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 (204)

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 (243) is called with `CoIndexError`.

See also:

For an example, see Pack (236).

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 (243) is called with `CoIndexError`. If the collection fails to expand, then `coOverFlow` is passd to `Error`.

See also: Insert (240)

```
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 (232).

Errors: Errors returned are those by TStream.Put (218).

See also: Load (232), PutItem (245)

For an example, see seeplLoadTCollection.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 (218).

See also: Store (245), GetItem (235).

11.11 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 (247) method.

Because the collection is ordered, `TSortedCollection` overrides some `TCollection` methods, to provide faster routines for lookup.

The Compare (247) 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 constuctor (see TCollection.Init (232)) 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 (247), Done (233)

For an example, see

TSortedCollection.Load

Declaration: `Constructor Load (Var S: TStream);`

Description: `Load` calls the inherited constuctor (see TCollection.Load (232)) 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 (246), Done (233)

For an example, see TCollection.Load (232).

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 (247), Compare (247).

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 (248), Compare (247).

For an example, see TCollection.IndexOf (234)

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 (248) method and in the Insert (249) 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 (247), Search (248)

```
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 (247) function to implement this search.

If the item is found, `Search` returns `True`, otherwise `False` is returned.

Errors: None.

See also: IndexOf (234).

```
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 (244), 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 (244)

```
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 (245), and then writing the `Duplicates` flag to the stream.

After a `Store`, the collection can be loaded from the stream with the constructor Load (247)

Errors: Errors can be those of TStream.Put (218).

See also: Load (247)

For an example, see TCollection.Load (232).

11.12 TStringCollection

The `TStringCollection` object manages a sorted collection of pascal strings. To this end, it overrides the Compare (247) 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 (216).

See also: PutItem (252)

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:

-1if the first string is alphabetically earlier than the second string.

0if the two strings are equal.

1if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare (247)

```
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 (241)

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.WriteStr (218).

See also: GetItem (251)

11.13 TStrCollection

The `TStrCollection` object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the Compare (247) 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 (214).

See also: PutItem (254)

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

-1if the first string is alphabetically earlier than the second string.

0if the two strings are equal.

1if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare (247)

```
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(I),At(I+1))=0 then
        Writeln ('Duplicate string found at position ',I);
  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 (241)

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 (218).

See also: GetItem (253)

11.14 TUnSortedStrCollection

The `TUnSortedStrCollection` object manages an unsorted list of objects. To this end, it overrides the TSortedCollection.Insert (249) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the Search (248) and IndexOf (234) 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 it's 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.
```

11.15 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 assiciated 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 (251)

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 manly for internal use by the TCollection.Load (232) method.

Errors: Errors returned are those by TStream.Read (219)

See also: TCollection.Load (232), TStream.Read (219)

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 resourceitem record.

It does NOT remove the item from the collection.

Errors: None.

See also: TCollection.FreeItem (241)

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 (245) method.

Errors: Errors returned are those by TStream.Write (220).

See also: Store (245)

11.16 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 (257) constructor call.

Modified is set to `True` if one of the resources has been changed. It is set by the SwitchTo (257), Delete (259) and Put (259) methods. Calling Flush (259) 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 (258)

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 (257)

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 (257)

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 (258)

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 (219) and TStream.Write (220).

See also: Flush (259)

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 (219) and TStream.Write (220).

See also: SwitchTo (258)

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 (259)

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 (218) and `TStream.Seek`

See also: Get (258)

11.17 TStringList

```
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;
```

11.18 TStrListMaker

```
TYPE
   TStrListMaker = OBJECT (TObject)
```

```
    Constructor Init (AStrSize, AIndexSize: Sw_Word);
    Destructor Done; Virtual;
    Procedure Put (Key: Sw_Word; S: String);
    Procedure Store (Var S: TStream);
 END;
 PStrListMaker = ^TStrListMaker;
```

Chapter 12

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. Its basic functionality is the same for both systems. 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.

12.1 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.

12.2 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 13

The SOCKETS unit.

This chapter describes the SOCKETS unit for Free Pascal. it was written for LINUX by Michaël Van Canneyt. 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.

13.1 Types, Constants and variables :

The following constants identify the different socket types, as needed in the Socket (273) 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 (273) 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 (273) 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 eddresses for the Bind (266), Recv (271) and Send (271) 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 (273) call.

```
TSockArray = Array[1..2] of Longint;
```

13.2 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. The accepted 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: 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_EOPNOTSUPP The socket type doesn't support the `Listen` operation.

SYS_EFAULT`Addr` points outside your address space.

SYS_EWOULDBLOCK The requested operation would block the process.

See also: Listen (270), Connect (267)

```
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 : ');
  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 (264) command. It is equivalent to subsequently calling the regular Accept (264) function and the Sock2Text (273) function. The function returns `True` if successfull, `False` otherwise.

Errors: The errors are those of Accept (264).

See also: Accept (264)

Accept

Declaration: `Function Accept (Sock:longint;var addr:string;var SockIn,SockOut:File) : Boolean;`

Description: This is an alternate form of the Accept (264) command. It is equivalent to subsequently calling the regular Accept (264) function and the Sock2File (272) function. The `Addr` parameter contains the name of the unix socket file to be opened. The function returns `True` if successfull, `False` otherwise.

Errors: The errors are those of Accept (264).

See also: Accept (264)

Accept

Declaration: `Function Accept (Sock:longint;var addr:TInetSockAddr;var SockIn,SockOut:File) : Boolean;`

Description: This is an alternate form of the Accept (264) command. It is equivalent to subsequently calling the regular Accept (264) function and the Sock2File (272) function. The `Addr` parameter contains the parameters of the internet socket that should be opened. The function returns `True` if successfull, `False` otherwise.

Errors: The errors are those of Accept (264).

See also: Accept (264)

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 succesful, `False` if not.

Errors: Errors are returned in `SocketError` and include the following:

SYS_EBADF The socket descriptor is invalid.

SYS_EINVAL The socket is already bound to an address,

SYS_EACCESS Address is protected and you don't have permission to open it.

More arrors can be found in the Unix man pages.

See also: Socket (273)

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 (??) and the regular Bind (266) function. The function returns `True` if successfull, `False` otherwise.

Errors: Errors are those of the regular Bind (266) command.

See also: Bind (266)

Connect

Declaration: `Function Connect (Sock:Longint;Var Addr;Addrlen:Longint) : Boolean;`

Description: `Connect` opens a connection to a peer, whose address is described by varAddr. `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`.

Errors: Errors are reported in `SocketError`.

See also: Listen (270), Bind (266), Accept (264)

```
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 (267) command. It is equivalent to subsequently calling the regular Connect (267) function and the Sock2Text (273) function. The function returns `True` if successfull, `False` otherwise.

Errors: The errors are those of Connect (267).

See also: Connect (267)

Connect

Declaration: `Function Connect (Sock:longint;const addr:string;var SockIn,SockOut:file) : Boolean;`

Description: This is an alternate form of the Connect (267) command. The parameter `addr` contains the name of the unix socket file to be opened. It is equivalent to subsequently calling the regular Connect (267) function and the Sock2File (272) function. The function returns `True` if successfull, `False` otherwise.

Errors: The errors are those of Connect (267).

See also: Connect (267)

Connect

Declaration: `Function Connect (Sock:longint;const addr: TInetSockAddr;var SockIn,SockOut:file) : Boolean;`

Description: This is another alternate form of the Connect (267) command. It is equivalent to subsequently calling the regular Connect (267) function and the Sock2File (272) function. The `Addr` parameter contains the parameters of the internet socket to connect to. The function returns `True` if successfull, `False` otherwise.

Errors: The errors are those of Connect (267).

See also: Connect (267)

```
program pfinger;

uses sockets,errors;
```

```
Var Addr : TInetSockAddr;
    S : Longint;
    Sin,Sout : Text;
    Line : string;

begin
  Addr.family:=AF_INET;
  { port 78 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;
  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 succesfully, `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_EBADF The socket descriptor is invalid.

SYS_ENOBUFS The system doesn't have enough buffers to perform the operation.

SYS_ENOTSOCK The descriptor is not a socket.

SYS_EFAULT`Addr` points outside your address space.

SYS_ENOTCONN The socket isn't connected.

See also: Connect (267), Socket (273), `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 succesfully, `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_EBADF The socket descriptor is invalid.

SYS_ENOBUFS The system doesn't have enough buffers to perform the operation.

SYS_ENOTSOCK The descriptor is not a socket.

SYS_EFAULT`Addr` points outside your address space.

See also: Bind (266)

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_SOCKET From the socket itself.

XXXset `Level` to `XXX`, the protocol number of the protocol which should interprete 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_EBADF The socket descriptor is invalid.

SYS_ENOTSOCK The descriptor is not a socket.

SYS_EFAULT`OptVal` points outside your address space.

See also: GetSocketOptions (270)

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_EBADF The socket descriptor is invalid.

SYS_ENOTSOCK The descriptor is not a socket.

SYS_EOPNOTSUPP The socket type doesn't support the `Listen` operation.

See also: Socket (273), Bind (266), Connect (267)

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 functions 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 (271)

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 (271), `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_SOCKET To manipulate the socket itself.

XXXset `Level` to `XXX`, the protocol number of the protocol which should interprete 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_EBADF The socket descriptor is invalid.

SYS_ENOTSOCK The descriptor is not a socket.

SYS_EFAULT`OptVal` points outside your address space.

See also: GetSocketOptions (270)

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_EBADF The socket descriptor is invalid.

SYS_ENOTCONN The socket isn't connected.

SYS_ENOTSOCK The descriptor is not a socket.

See also: Socket (273), Connect (267)

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 (273), Sock2Text (273)

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 (273), Sock2File (272)

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 (266) call. If unsuccesfull, the function returns -1.

Errors: Errors are returned in `SocketError`, and include the follwing:

SYS_EPROTONOSUPPORT The protocol type or the specified protocol is not supported within this domain.

SYS_EMFILE The per-process descriptor table is full.

SYS_ENFILE The system file table is full.

SYS_EACCESS Permission to create a socket of the specified type and/or protocol is denied.

SYS_ENOBUFS Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

See also: SocketPair (273), `socket` (2)

for an example, see Accept (264).

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 (273)

See also: Str2UnixSockAddr(const addr:string;var t:TUnixSockAddr;var len:longint)

`Str2UnixSockAddr` transforms a Unix socket address in a string to a `TUnixSockAddr` sturcture which can be passed to the Bind (266) call. None. Socket (273), Bind (266)

Chapter 14

The STRINGS unit.

This chapter describes the `STRINGS` unit for Free Pascal. 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.

14.1 Functions and procedures.

StrAlloc

Declaration: `Procedure StrAlloc (Len : Longint);`

Description: PChar

Errors: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

See also: StrPCopy.

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 (278), StrIComp (277), StrLIComp (280)

For an example, see StrLComp (278).

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 (282), StrLCopy (279), StrECopy (276)

```
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 (281)

```
Program Example17;

Uses strings;

{ Program to demonstrate the StrDispose function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  Writeln ('Before StnNew : 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 (279), StrCopy (275)

```
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 (279)

```
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 (278), StrComp (275), StrLIComp (280)

```
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.
```

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 (274)

```
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 (275), StrIComp (277), StrLIComp (280)

```
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 (275), StrECopy (276)

```
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.
```

StrLIComp

Declaration: `Function StrLIComp (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 (278), StrComp (275), StrIComp (277)

For an example, see StrIComp (277)

StrLower

Declaration: `Function StrLower (P : PChar) : PChar;`

Description: Converts `P` to an all-lowercase string. Returns `P`.

Errors: None.

See also: `Upcase` () , StrUpper (284)

```
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
  Writeln ('Uppercase : ',StrUpper(P2));
  StrLower (P1);
  Writeln ('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 (279), StrCopy (275)

```
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 (275), StrDispose (276)

```
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 (282)

```
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 (282)

```
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 (283), StrRScan (283)

```
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 (283), StrPos (283)

For an example, see StrScan (283).

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 (283), StrPos (283)

```
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: `Upcase` () , StrLower (280)

For an example, see StrLower (280)

Index