Run-Time Library (RTL) : Reference guide.

Free Pascal version 2.2:
Reference guide for RTL units.
Document version 2.1
October 2008

Michaël Van Canneyt

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 the Free Pascal Run-Time library (RTL).

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.

0.1 Overview

The Run-Time Library is the basis of all Free Pascal programs. It contains the basic units that most programs will use, and are made available on all platforms supported by Free pascal (well, more or less).

There are units for compatibility with the Turbo Pascal Run-Time library, and there are units for compatibility with Delphi.

On top of these two sets, there are also a series of units to handle keyboard/mouse and text screens in a cross-platform way.

Other units include platform specific units that implement the specifics of a platform, these are usually needed to support the Turbo Pascal or Delphi units.

Units that fall outside the above outline do not belong in the RTL, but should be included in the packages, or in the FCL.

Chapter 1

Reference for unit 'BaseUnix'

1.1 Used units

Table 1.1: Used units by unit 'BaseUnix'

Name	Page
unixtype	1575

1.2 Overview

The BaseUnix unit was implemented by Marco Van de Voort. It contains basic unix functionality. It supersedes the Linux unit of version 1.0.X of the compiler, but does not implement all functionality of the linux unit.

People that have code which heavily uses the old Linux unit, can simply change linux by oldlinux in the `uses` clause of their projects, but they should really consider moving to the Unix and BaseUnix units.

For porting FPC to new unix-like platforms, it should be sufficient to implement the functionality in this unit for the new platform.

1.3 Constants, types and variables

1.3.1 Constants

```
ARG_MAX = UnixType.ARG_MAX
```

Maximum number of arguments to a program.

```
BITSINWORD = 8 * sizeof ( cuLong )
```

Number of bits in a word.

```
ESysE2BIG = 7
```

System error: Argument list too long

```
ESysEACCES = 13
```

System error: Permission denied

```
ESysEADDRINUSE = 98
```

System error: Address already in use

```
ESysEADDRNOTAVAIL = 99
```

System error: Cannot assign requested address

```
ESysEADV = 68
```

System error: Advertise error

```
ESysEAFNOSUPPORT = 97
```

System error: Address family not supported by protocol

```
ESysEAGAIN = 11
```

System error: Try again

```
ESysEALREADY = 114
```

System error: Operation already in progress

```
ESysEBADE = 52
```

System error: Invalid exchange

```
ESysEBADF = 9
```

System error: Bad file number

```
ESysEBADFD = 77
```

System error: File descriptor in bad state

```
ESysEBADMSG = 74
```

System error: Not a data message

```
ESysEBADR = 53
```

System error: Invalid request descriptor

```
ESysEBADRQC = 56
```

System error: Invalid request code

```
ESysEBADSLT = 57
```

System error: Invalid slot

```
ESysEBFONT = 59
```

System error: Bad font file format

```
ESysEBUSY = 16
```

System error: Device or resource busy

```
ESysECHILD = 10
```

System error: No child processes

```
ESysECHRNG = 44
```

System error: Channel number out of range

```
ESysECOMM = 70
```

System error: Communication error on send

```
ESysECONNABORTED = 103
```

System error: Software caused connection abort

```
ESysECONNREFUSED = 111
```

System error: Connection refused

```
ESysECONNRESET = 104
```

System error: Connection reset by peer

```
ESysEDEADLK = 35
```

System error: Resource deadlock would occur

```
ESysEDEADLOCK = 58
```

System error: File locking deadlock error

```
ESysEDESTADDRREQ = 89
```

System error: Destination address required

```
ESysEDOM = 33
```

System error: Math argument out of domain of func

```
ESysEDOTDOT = 73
```

System error: RFS specific error

```
ESysEDQUOT = 122
```

System error: Quota exceeded

```
ESysEEXIST = 17
```

System error: File exists

```
ESysEFAULT = 14
```

System error: Bad address

```
ESysEFBIG = 27
```

System error: File too large

```
ESysEHOSTDOWN = 112
```

System error: Host is down

```
ESysEHOSTUNREACH = 113
```

System error: No route to host

```
ESysEIDRM = 43
```

System error: Identifier removed

```
ESysEILSEQ = 84
```

System error: Illegal byte sequence

```
ESysEINPROGRESS = 115
```

System error: Operation now in progress

```
ESysEINTR = 4
```

System error: Interrupted system call

```
ESysEINVAL = 22
```

System error: Invalid argument

```
ESysEIO = 5
```

System error: I/O error

```
ESysEISCONN = 106
```

System error: Transport endpoint is already connected

```
ESysEISDIR = 21
```

System error: Is a directory

```
ESysEISNAM = 120
```

System error: Is a named type file

```
ESysEL2HLT = 51
```

System error: Level 2 halted

```
ESysEL2NSYNC = 45
```

System error: Level 2 not synchronized

```
ESysEL3HLT = 46
```

System error: Level 3 halted

```
ESysEL3RST = 47
```

System error: Level 3 reset

```
ESysELIBACC = 79
```

System error: Can not access a needed shared library

```
ESysELIBBAD = 80
```

System error: Accessing a corrupted shared library

```
ESysELIBEXEC = 83
```

System error: Cannot exec a shared library directly

```
ESysELIBMAX = 82
```

System error: Attempting to link in too many shared libraries

```
ESysELIBSCN = 81
```

System error: .lib section in a.out corrupted

```
ESysELNRNG = 48
```

System error: Link number out of range

```
ESysELOOP = 40
```

System error: Too many symbolic links encountered

```
ESysEMFILE = 24
```

System error: Too many open files

```
ESysEMLINK = 31
```

System error: Too many links

```
ESysEMSGSIZE = 90
```

System error: Message too long

```
ESysEMULTIHOP = 72
```

System error: Multihop attempted

```
ESysENAMETOOLONG = 36
```

System error: File name too long

```
ESysENAVAIL = 119
```

System error: No XENIX semaphores available

```
ESysENETDOWN = 100
```

System error: Network is down

```
ESysENETRESET = 102
```

System error: Network dropped connection because of reset

```
ESysENETUNREACH = 101
```

System error: Network is unreachable

```
ESysENFILE = 23
```

System error: File table overflow

```
ESysENOANO = 55
```

System error: No anode

```
ESysENOBUFS = 105
```

System error: No buffer space available

```
ESysENOCSI = 50
```

System error: No CSI structure available

```
ESysENODATA = 61
```

System error: No data available

```
ESysENODEV = 19
```

System error: No such device

```
ESysENOENT = 2
```

System error: No such file or directory

```
ESysENOEXEC = 8
```

System error: Exec format error

```
ESysENOLCK = 37
```

System error: No record locks available

```
ESysENOLINK = 67
```

System error: Link has been severed

```
ESysENOMEM = 12
```

System error: Out of memory

```
ESysENOMSG = 42
```

System error: No message of desired type

```
ESysENONET = 64
```

System error: Machine is not on the network

```
ESysENOPKG = 65
```

System error: Package not installed

```
ESysENOPROTOOPT = 92
```

System error: Protocol not available

```
ESysENOSPC = 28
```

System error: No space left on device

```
ESysENOSR = 63
```

System error: Out of streams resources

```
ESysENOSTR = 60
```

System error: Device not a stream

```
ESysENOSYS = 38
```

System error: Function not implemented

```
ESysENOTBLK = 15
```

System error: Block device required

```
ESysENOTCONN = 107
```

System error: Transport endpoint is not connected

```
ESysENOTDIR = 20
```

System error: Not a directory

```
ESysENOTEMPTY = 39
```

System error: Directory not empty

```
ESysENOTNAM = 118
```

System error: Not a XENIX named type file

```
ESysENOTSOCK = 88
```

System error: Socket operation on non-socket

```
ESysENOTTY = 25
```

System error: Not a typewriter

```
ESysENOTUNIQ = 76
```

System error: Name not unique on network

```
ESysENXIO = 6
```

System error: No such device or address

```
ESysEOPNOTSUPP = 95
```

System error: Operation not supported on transport endpoint

```
ESysEOVERFLOW = 75
```

System error: Value too large for defined data type

```
ESysEPERM = 1
```

System error: Operation not permitted.

```
ESysEPFNOSUPPORT = 96
```

System error: Protocol family not supported

```
ESysEPIPE = 32
```

System error: Broken pipe

```
ESysEPROTO = 71
```

System error: Protocol error

```
ESysEPROTONOSUPPORT = 93
```

System error: Protocol not supported

```
ESysEPROTOTYPE = 91
```

System error: Protocol wrong type for socket

```
ESysERANGE = 34
```

System error: Math result not representable

```
ESysEREMCHG = 78
```

System error: Remote address changed

```
ESysEREMOTE = 66
```

System error: Object is remote

```
ESysEREMOTEIO = 121
```

System error: Remote I/O error

```
ESysERESTART = 85
```

System error: Interrupted system call should be restarted

```
ESysEROFS = 30
```

System error: Read-only file system

```
ESysESHUTDOWN = 108
```

System error: Cannot send after transport endpoint shutdown

```
ESysESOCKTNOSUPPORT = 94
```

System error: Socket type not supported

```
ESysESPIPE = 29
```

System error: Illegal seek

```
ESysESRCH = 3
```

System error: No such process

```
ESysESRMNT = 69
```

System error: Srmount error

```
ESysESTALE = 116
```

System error: Stale NFS file handle

```
ESysESTRPIPE = 86
```

System error: Streams pipe error

```
ESysETIME = 62
```

System error: Timer expired

```
ESysETIMEDOUT = 110
```

System error: Connection timed out

```
ESysETOOMANYREFS = 109
```

System error: Too many references: cannot splice

```
ESysETXTBSY = 26
```

System error: Text (code segment) file busy

```
ESysEUCLEAN = 117
```

System error: Structure needs cleaning

```
ESysEUNATCH = 49
```

System error: Protocol driver not attached

```
ESysEUSERS = 87
```

System error: Too many users

```
ESysEWOULDBLOCK = ESysEAGAIN
```

System error: Operation would block

```
ESysEXDEV = 18
```

System error: Cross-device link

```
ESysEXFULL = 54
```

System error: Exchange full

```
FD_MAXFDSET = 1024
```

Maximum elements in a TFDSet (126) array.

```
FPE_FLTDIV = 3
```

Value signalling floating point divide by zero in case of SIGFPE signal

```
FPE_FLTINV = 7
```

Value signalling floating point invalid operation in case of SIGFPE signal

```
FPE_FLTOVF = 4
```

Value signalling floating point overflow in case of SIGFPE signal

```
FPE_FLTRES = 6
```

Value signalling floating point inexact result in case of SIGFPE signal

```
FPE_FLTSUB = 8
```

Value signalling floating point subscript out of range in case of SIGFPE signal

```
FPE_FLTUND = 5
```

Value signalling floating point underflow in case of SIGFPE signal

```
FPE_INTDIV = 1
```

Value signalling integer divide in case of SIGFPE signal

```
FPE_INTOVF = 2
```

Value signalling integer overflow in case of SIGFPE signal

```
F_GetFd = 1
```

fpFCntl (141) command: Get close-on-exec flag

```
F_GetFl = 3
```

fpFCntl (141) command: Get filedescriptor flags

```
F_GetLk = 5
```

fpFCntl (141) command: Get lock

```
F_GetOwn = 9
```

fpFCntl (141) command: get owner of filedescriptor events

```
F_OK = 0
```

fpAccess (132) call test: file exists.

```
F_SetFd = 2
```

fpFCntl (141) command: Set close-on-exec flag

```
F_SetFl = 4
```

fpFCntl (141) command: Set filedescriptor flags

```
F_SetLk = 6
```

fpFCntl (141) command: Set lock

```
F_SetLkW = 7
```

fpFCntl (141) command: Test lock

```
F_SetOwn = 8
```

fpFCntl (141) command: Set owner of filedescriptor events

```
ln2bitmask = 1 shl ln2bitsinword - 1
```

Last bit in word.

```
MAP_ANON = MAP_ANONYMOUS
```

Anonymous memory mapping (data private to application)

```
MAP_ANONYMOUS = $20
```

FpMMap (154) map type: Don't use a file

```
MAP_FAILED = pointer ( - 1 )
```

Memory mapping failed error code

```
MAP_FIXED = $10
```

FpMMap (154) map type: Interpret addr exactly

```
MAP_PRIVATE = $2
```

FpMMap (154) map type: Changes are private

```
MAP_SHARED = $1
```

FpMMap (154) map type: Share changes

```
MAP_TYPE = $f
```

FpMMap (154) map type: Bitmask for type of mapping

```
NAME_MAX = UnixType.NAME_MAX
```

Maximum filename length.

```
O_APPEND = $400
```

fpOpen (157) file open mode: Append to file

```
O_CREAT = $40
```

fpOpen (157) file open mode: Create if file does not yet exist.

```
O_DIRECT = $4000
```

fpOpen (157) file open mode: Minimize caching effects

```
O_DIRECTORY = $10000
```

fpOpen (157) file open mode: File must be directory.

```
O_EXCL = $80
```

fpOpen (157) file open mode: Open exclusively

```
O_NDELAY = O_NONBLOCK
```

fpOpen (157) file open mode: Alias for O_NonBlock (107)

```
O_NOCTTY = $100
```

fpOpen (157) file open mode: No TTY control.

```
O_NOFOLLOW = $20000
```

fpOpen (157) file open mode: Fail if file is symbolic link.

```
O_NONBLOCK = $800
```

fpOpen (157) file open mode: Open in non-blocking mode

```
O_RDONLY = 0
```

fpOpen (157) file open mode: Read only

```
O_RDWR = 2
```

fpOpen (157) file open mode: Read/Write

```
O_SYNC = $1000
```

fpOpen (157) file open mode: Write to disc at once

```
O_TRUNC = $200
```

fpOpen (157) file open mode: Truncate file to length 0

```
O_WRONLY = 1
```

fpOpen (157) file open mode: Write only

```
PATH_MAX = UnixType.PATH_MAX
```

Maximum pathname length.

```
PRIO_PGRP = UnixType.PRIO_PGRP
```

Easy access alias for unixtype.PRIO_PGRP (1575)

```
PRIO_PROCESS = UnixType.PRIO_PROCESS
```

Easy access alias for unixtype.PRIO_PROCESS (1575)

```
PRIO_USER = UnixType.PRIO_USER
```

Easy access alias for unixtype.PRIO_USER (1575)

```
PROT_EXEC = $4
```

FpMMap (154) memory access: page can be executed

```
PROT_NONE = $0
```

FpMMap (154) memory access: page can not be accessed

```
PROT_READ = $1
```

FpMMap (154) memory access: page can be read

```
PROT_WRITE = $2
```

FpMMap (154) memory access: page can be written

```
RLIMIT_AS = 9
```

RLimit request address space limit

```
RLIMIT_CORE = 4
```

RLimit request max core file size

```
RLIMIT_CPU = 0
```

RLimit request CPU time in ms

```
RLIMIT_DATA = 2
```

RLimit request max data size

```
RLIMIT_FSIZE = 1
```

Rlimit request maximum filesize

```
RLIMIT_LOCKS = 10
```

RLimit request maximum file locks held

```
RLIMIT_MEMLOCK = 8
```

RLimit request max locked-in-memory address space

```
RLIMIT_NOFILE = 7
```

RLimit request max number of open files

```
RLIMIT_NPROC = 6
```

RLimit request max number of processes

```
RLIMIT_RSS = 5
```

RLimit request max resident set size

```
RLIMIT_STACK = 3
```

RLimit request max stack size

```
R_OK = 4
```

fpAccess (132) call test: read allowed

```
SA_INTERRUPT = $20000000
```

Sigaction options: ?

```
SA_NOCLDSTOP = 1
```

Sigaction options: Do not receive notification when child processes stop

```
SA_NOCLDWAIT = 2
```

Sigaction options: ?

```
SA_NODEFER = $40000000
```

Sigaction options: Do not mask signal in its own signal handler

```
SA_NOMASK = SA_NODEFER
```

Sigaction options: Do not prevent the signal from being received when it is handled.

```
SA_ONESHOT = SA_RESETHAND
```

Sigaction options: Restore the signal action to the default state.

```
SA_ONSTACK = $08000000
```

SA_ONSTACK is used in the fpsigaction (168) to indicate the signal handler must be called on an alternate signal stack provided by fpsigaltstack (94). If an alternate stack is not available, the default stack will be used.

```
SA_RESETHAND = $80000000
```

Sigaction options: Restore signal action to default state when signal handler exits.

```
SA_RESTART = $10000000
```

Sigaction options: Provide behaviour compatible with BSD signal semantics

```
SA_RESTORER = $04000000
```

Signal restorer handler

```
SA_SIGINFO = 4
```

Sigaction options: The signal handler takes 3 arguments, not one.

```
SEEK_CUR = 1
```

fpLSeek (151) option: Set position relative to current position.

```
SEEK_END = 2
```

fpLSeek (151) option: Set position relative to end of file.

```
SEEK_SET = 0
```

fpLSeek (151) option: Set absolute position.

```
SIGABRT = 6
```

Signal: ABRT (Abort)

```
SIGALRM = 14
```

Signal: ALRM (Alarm clock)

```
SIGBUS = 7
```

Signal: BUS (bus error)

```
SIGCHLD = 17
```

Signal: CHLD (child status changed)

```
SIGCONT = 18
```

Signal: CONT (Continue)

```
SIGFPE = 8
```

Signal: FPE (Floating point error)

```
SIGHUP = 1
```

Signal: HUP (Hangup)

```
SIGILL = 4
```

Signal: ILL (Illegal instruction)

```
SIGINT = 2
```

Signal: INT (Interrupt)

```
SIGIO = 29
```

Signal: IO (I/O operation possible)

```
SIGIOT = 6
```

Signal: IOT (IOT trap)

```
SIGKILL = 9
```

Signal: KILL (unblockable)

```
SIGPIPE = 13
```

Signal: PIPE (Broken pipe

```
SIGPOLL = SIGIO
```

Signal: POLL (Pollable event)

```
SIGPROF = 27
```

Signal: PROF (Profiling alarm)

```
SIGPWR = 30
```

Signal: PWR (power failure restart)

```
SIGQUIT = 3
```

Signal: QUIT

```
SIGSEGV = 11
```

Signal: SEGV (Segmentation violation)

```
SIGSTKFLT = 16
```

Signal: STKFLT (Stack Fault)

```
SIGSTOP = 19
```

Signal: STOP (Stop, unblockable)

```
SIGTerm = 15
```

Signal: TERM (Terminate)

```
SIGTRAP = 5
```

Signal: TRAP (Trace trap)

```
SIGTSTP = 20
```

Signal: TSTP (keyboard stop)

```
SIGTTIN = 21
```

Signal: TTIN (Terminal input, background)

```
SIGTTOU = 22
```

Signal: TTOU (Terminal output, background)

```
SIGUNUSED = 31
```

Signal: Unused

```
SIGURG = 23
```

Signal: URG (Socket urgent condition)

```
SIGUSR1 = 10
```

Signal: USR1 (User-defined signal 1)

```
SIGUSR2 = 12
```

Signal: USR2 (User-defined signal 2)

```
SIGVTALRM = 26
```

Signal: VTALRM (Virtual alarm clock)

```
SIGWINCH = 28
```

Signal: WINCH (Window/Terminal size change)

```
SIGXCPU = 24
```

Signal: XCPU (CPU limit exceeded)

```
SIGXFSZ = 25
```

Signal: XFSZ (File size limit exceeded)

```
SIG_BLOCK = 0
```

Sigprocmask flags: Add signals to the set of blocked signals.

```
SIG_DFL = 0
```

Signal handler: Default signal handler

```
SIG_ERR = -1
```

Signal handler: error

```
SIG_IGN = 1
```

Signal handler: Ignore signal

```
SIG_MAXSIG = UnixType.SIG_MAXSIG
```

Maximum system signal number.

```
SIG_SETMASK = 2
```

Sigprocmask flags: Set of blocked signals is given.

```
SIG_UNBLOCK = 1
```

Sigprocmask flags: Remove signals from the set set of blocked signals.

```
SI_PAD_SIZE = ( ( 128 div sizeof ( longint ) ) - 3 )
```

Signal information pad size.

```
SYS_NMLN = UnixType.SYS_NMLN
```

Max system name length.

```
S_IFBLK = 24576
```

File (#rtl.baseunix.stat (126) record) mode: Block device

```
S_IFCHR = 8192
```

File (#rtl.baseunix.stat (126) record) mode: Character device

```
S_IFDIR = 16384
```

File (#rtl.baseunix.stat (126) record) mode: Directory

```
S_IFIFO = 4096
```

File (#rtl.baseunix.stat (126) record) mode: FIFO

```
S_IFLNK = 40960
```

File (#rtl.baseunix.stat (126) record) mode: Link

```
S_IFMT = 61440
```

File (#rtl.baseunix.stat (126) record) mode: File type bit mask

```
S_IFREG = 32768
```

File (#rtl.baseunix.stat (126) record) mode: Regular file

```
S_IFSOCK = 49152
```

File (#rtl.baseunix.stat (126) record) mode: Socket

```
S_IRGRP = %0000100000
```

Mode flag: Read by group.

```
S_IROTH = %0000000100
```

Mode flag: Read by others.

```
S_IRUSR = %0100000000
```

Mode flag: Read by owner.

```
S_IRWXU = S_IRUSR or S_IWUSR or S_IXUSR
```

Mode flag: Read, write, execute by user.

```
S_IWGRP = %0000010000
```

Mode flag: Write by group.

```
S_IWOTH = %0000000010
```

Mode flag: Write by others.

```
S_IWUSR = %0010000000
```

Mode flag: Write by owner.

```
S_IXGRP = %0000001000
```

Mode flag: Execute by group.

```
S_IXOTH = %0000000001
```

Mode flag: Execute by others.

```
S_IXUSR = %0001000000
```

Mode flag: Execute by owner.

```
UTSNAME_DOMAIN_LENGTH = UTSNAME_LENGTH
```

Max length of utsname (131) domain name.

```
UTSNAME_LENGTH = SYS_NMLN
```

Max length of utsname (131) system name, release, version, machine.

```
UTSNAME_NODENAME_LENGTH = UTSNAME_LENGTH
```

Max length of utsname (131) node name.

```
WNOHANG = 1
```

#rtl.baseunix.fpWaitpid (181) option: Do not wait for processes to terminate.

```
wordsinfdset = FD_MAXFDSET div BITSINWORD
```

Number of words in a TFDSet (126) array

```
wordsinsigset = SIG_MAXSIG div BITSINWORD
```

Number of words in a signal set.

```
WUNTRACED = 2
```

#rtl.baseunix.fpWaitpid (181) option: Also report children wich were stopped but not yet reported

```
W_OK = 2
```

fpAccess (132) call test: write allowed

```
X_OK = 1
```

fpAccess (132) call test: execute allowed

```
_STAT_VER = _STAT_VER_LINUX
```

Stat version number

```
_STAT_VER_KERNEL = 1
```

Current version of stat record

```
_STAT_VER_LINUX = 3
```

Version of linux stat record

```
_STAT_VER_LINUX_OLD = 1
```

Old kernel definition of stat

```
_STAT_VER_SVR4 = 2
```

SVR 4 definition of stat

1.3.2 Types

```
Blkcnt64_t = cuint64
```

64-bit block count

```
Blkcnt_t = cuint
```

Block count type.

```
Blksize_t = cuint
```

Block size type.

```
cbool = UnixType.cbool
```

Boolean type

```
cchar = UnixType.cchar
```

Alias for #rtl.UnixType.cchar (1577)

```
cdouble = UnixType.cdouble
```

Double precision real format.

```
cfloat = UnixType.cfloat
```

Floating-point real format

```
cint = UnixType.cint
```

C type: integer (natural size)

```
cint16 = UnixType.cint16
```

C type: 16 bits sized, signed integer.

```
cint32 = UnixType.cint32
```

C type: 32 bits sized, signed integer.

```
cint64 = UnixType.cint64
```

C type: 64 bits sized, signed integer.

```
cint8 = UnixType.cint8
```

C type: 8 bits sized, signed integer.

```
clock_t = UnixType.clock_t
```

Clock ticks type

`clong = UnixType.clong`

C type: long signed integer (double sized)

`clongdouble = UnixType.clongdouble`

Usually translates to an extended, but is CPU dependent.

`clonglong = UnixType.clonglong`

C type: 64-bit (double long) signed integer.

`cschar = UnixType.cschar`

Signed character type

`cshort = UnixType.cshort`

C type: short signed integer (half sized)

`csigned = UnixType.csigned`

`csigned` is an alias for cint (116).

`csint = UnixType.csint`

Signed integer

`cslong = UnixType.cslong`

The size is CPU dependent.

`cslonglong = UnixType.cslonglong`

`cslonglong` is an alias for clonglong (117).

`csshort = UnixType.csshort`

Short signed integer type

`cuchar = UnixType.cuchar`

Alias for #rtl.UnixType.cuchar (1578)

`cuint = UnixType.cuint`

C type: unsigned integer (natural size)

`cuint16 = UnixType.cuint16`

C type: 16 bits sized, unsigned integer.

```
cuint32 = UnixType.cuint32
```

C type: 32 bits sized, unsigned integer.

```
cuint64 = UnixType.cuint64
```

C type: 64 bits sized, unsigned integer.

```
cuint8 = UnixType.cuint8
```

C type: 8 bits sized, unsigned integer.

```
culong = UnixType.culong
```

C type: long unsigned integer (double sized)

```
culonglong = UnixType.culonglong
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = UnixType.cunsigned
```

Alias for #rtl.unixtype.cunsigned (1579)

```
cushort = UnixType.cushort
```

C type: short unsigned integer (half sized)

```
dev_t = UnixType.dev_t
```

Device descriptor type.

```
Dir = record
  dd_fd : Integer;
  dd_loc : LongInt;
  dd_size : Integer;
  dd_buf : pDirent;
  dd_nextoff : cardinal;
  dd_max : Integer;
  dd_lock : pointer;
end
```

Record used in fpOpenDir (158) and fpReadDir (162) calls

```
Dirent = packed record
  d_fileno : ino64_t;
  d_off : off_t;
  d_reclen : cushort;
  d_type : cuchar;
  d_name : Array[0..(255+1)-1] of Char;
end
```

Record used in the fpReadDir (162) function to return files in a directory.

```
FLock = record
  l_type : cshort;
  l_whence : cshort;
  l_start : off_t;
  l_len : off_t;
  l_pid : pid_t;
end
```

Lock description type for fpFCntl (141) lock call.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
iovec = record
  iov_base : pointer;
  iov_len : size_t;
end
```

`iovec` is used in fpreadv (164) for IO to multiple buffers to describe a buffer location.

```
mode_t = UnixType.mode_t
```

Inode mode type.

```
nlink_t = UnixType.nlink_t
```

Number of links type.

```
off_t = UnixType.off_t
```

Offset type.

```
PBlkCnt = ^Blkcnt_t
```

pointer to TBlkCnt (126) type.

```
PBlkSize = ^Blksize_t
```

Pointer to TBlkSize (126) type.

```
pcbool = UnixType.pcbool
```

Pointer to boolean type cbool (116)

```
pcchar = UnixType.pcchar
```

Alias for #rtl.UnixType.pcchar (1580)

```
pcdouble = UnixType.pcdouble
```

Pointer to cdouble (116) type.

```
pcfloat = UnixType.pcfloat
```

Pointer to cfloat (116) type.

```
pcint = UnixType.pcint
```

Pointer to cInt (116) type.

```
pcint16 = UnixType.pcint16
```

Pointer to 16-bit signed integer type

```
pcint32 = UnixType.pcint32
```

Pointer to signed 32-bit integer type

```
pcint64 = UnixType.pcint64
```

Pointer to signed 64-bit integer type

```
pcint8 = UnixType.pcint8
```

Pointer to 8-bits signed integer type

```
pClock = UnixType.pClock
```

Pointer to TClock (126) type.

```
pclong = UnixType.pclong
```

Pointer to cLong (117) type.

```
pclongdouble = UnixType.pclongdouble
```

Pointer to the long double type clongdouble (117)

```
pclonglong = UnixType.pclonglong
```

Pointer to longlong type.

```
pcschar = UnixType.pcschar
```

Pointer to character type cschar (117).

```
pcshort = UnixType.pcshort
```

Pointer to cShort (117) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to signed integer type csigned (117).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type csint (117)

```
pcslong = UnixType.pcslong
```

Pointer ot the signed long cslong (117)

```
pcslonglong = UnixType.pcslonglong
```

Pointer to Signed longlong type cslonglong (117)

```
pcsshort = UnixType.pcsshort
```

Pointer to short signed integer type csshort (117)

```
pcuchar = UnixType.pcuchar
```

Alias for #rtl.UnixType.pcuchar (1581)

```
pcuint = UnixType.pcuint
```

Pointer to cUInt (117) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to 16-bit unsigned integer type

```
pcuint32 = UnixType.pcuint32
```

Pointer to unsigned 32-bit integer type

```
pcuint64 = UnixType.pcuint64
```

Pointer to unsigned 64-bit integer type

```
pcuint8 = UnixType.pcuint8
```

Pointer to 8-bits unsigned integer type

```
pculong = UnixType.pculong
```

Pointer to cuLong (118) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type

```
pcunsigned = UnixType.pcunsigned
```

Alias for #rtl.unixtype.pcunsigned (1582)

```
pcushort = UnixType.pcushort
```

Pointer to cuShort (118) type.

```
pDev = UnixType.pDev
```

Pointer to TDev (126) type.

```
pDir = ^Dir
```

Pointer to TDir (126) record

```
pDirent = ^Dirent
```

Pointer to TDirent (126) record.

```
pFDSet = ^TFDSet
```

Pointer to TFDSet (126) type.

```
pFilDes = ^TFilDes
```

Pointer to TFilDes (127) type.

```
pfpstate = ^tfpstate
```

Pointer to tfpstate (127) record.

```
pGid = UnixType.pGid
```

Pointer to TGid (127) type.

```
pGrpArr = ^TGrpArr
```

Pointer to TGrpArr (127) array.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (128) type.

```
piovec = ^tiovec
```

pointer to a iovec (119) record

```
pMode = UnixType.pMode
```

Pointer to TMode (128) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (128) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (128) type.

```
pPid = UnixType.pPid
```

Pointer to TPid (128) type.

```
PRLimit = ^TRLimit
```

Pointer to TRLimit (128) record

```
psigactionrec = ^sigactionrec
```

Pointer to SigActionRec (125) record type.

```
PSigContext = ^TSigContext
```

Pointer to #rtl.baseunix.TSigContext (129) record type.

```
psiginfo = ^tsiginfo
```

Pointer to #rtl.baseunix.TSigInfo (129) record type.

```
psigset = ^tsigset
```

Pointer to SigSet (126) type.

```
pSize = UnixType.pSize
```

Pointer to TSize (130) type.

```
pSize_t = UnixType.pSize_t
```

```
pSocklen = UnixType.pSocklen
```

Pointer to TSockLen (130) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (130) type

```
PStat = ^Stat
```

Pointer to TStat (130) type.

```
pthread_cond_t = UnixType.pthread_cond_t
```

Thread conditional variable type.

```
pthread_mutex_t = UnixType.pthread_mutex_t
```

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

Posix thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (130) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (127) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (127) type.

```
ptimezone = ^timezone
```

Pointer to TimeZone (127) record.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (128) type.

```
PTms = ^tms
```

Pointer to TTms (130) type.

```
Pucontext = ^Tucontext
```

Pointer to TUContext (131) type.

```
pUid = UnixType.pUid
```

Pointer to TUid (131) type.

```
pUtimBuf = ^UTimBuf
```

Pointer to TUTimBuf (131) type.

```
PUtsName = ^TUtsName
```

Pointer to TUtsName (131) type.

```
rlim_t = culong
```

rlim_t is used as the type for the various fields in the TRLimit (128) record.

```
sigactionhandler = sigactionhandler_t
```

When installing a signal handler, the actual signal handler must be of type SigActionHandler.

```
sigactionhandler_t = procedure(signal: LongInt;info: psiginfo;
                               context: PSigContext)
```

Standard signal action handler prototype

```
sigactionrec = record
  sa_handler : sigactionhandler_t;
  sa_flags : culong;
  sa_restorer : sigrestorerhandler_t;
  sa_mask : sigset_t;
end
```

Record used in fpSigAction (168) call.

```
signalhandler = signalhandler_t
```

Simple signal handler prototype

```
signalhandler_t = procedure(signal: LongInt)
```

Standard signal handler prototype

```
sigrestorerhandler = sigrestorerhandler_t
```

Alias for sigrestorerhandler_t (125) type.

```
sigrestorerhandler_t = procedure
```

Standard signal action restorer prototype

```
sigset = sigset_t
```

Signal set type

```
sigset_t = Array[0..wordsinsigset-1] of culong
```

Signal set type

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

```
Stat = packed record
end
```

Record describing an inode (file) in the FPFstat (143) call.

```
TBlkCnt = Blkcnt_t
```

Alias for Blkcnt_t (116) type.

```
TBlkSize = Blksize_t
```

Alias for blksize_t (116) type.

```
TClock = UnixType.TClock
```

Alias for clock_t (117) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (118) type.

```
TDir = Dir
```

Alias for Dir (118) type.

```
TDirent = Dirent
```

Alias for Dirent (119) type.

```
TFDSet = Array[0..(FD_MAXFDSETdivBITSINWORD)-1] of culong
```

File descriptor set for fpSelect (165) call.

```
TFilDes = Array[0..1] of cint
```

Array of file descriptors as used in fpPipe (160) call.

```
tfpreg = record
  significand : Array[0..3] of Word;
  exponent : Word;
end
```

Record describing floating point register in signal handler.

```
tfpstate = record
  cw : cardinal;
  sw : cardinal;
  tag : cardinal;
  ipoff : cardinal;
  cssel : cardinal;
  dataoff : cardinal;
  datasel : cardinal;
  st : Array[0..7] of tfpreg;
  status : cardinal;
end
```

Record describing floating point unit in signal handler.

```
TGid = UnixType.TGid
```

Alias for gid_t (119) type.

```
TGrpArr = Array[0..0] of TGid
```

Array of gid_t (119) IDs

```
timespec = UnixType.timespec
```

Short time specification type.

```
timeval = UnixType.timeval
```

Time specification type.

```
timezone = packed record
  tz_minuteswest : cint;
  tz_dsttime : cint;
end
```

Record describing a timezone

```
time_t = UnixType.time_t
```

Time span type

```
TIno = UnixType.TIno
```

Alias for ino_t (119) type.

```
TIOCtlRequest = UnixType.TIOCtlRequest
```

Easy access alias for unixtype.TIOCtlRequest (1587)

```
tiovec = iovec
```

Alias for the iovec (119) record type.

```
TMode = UnixType.TMode
```

Alias for mode_t (119) type.

```
tms = packed record
  tms_utime : clock_t;
  tms_stime : clock_t;
  tms_cutime : clock_t;
  tms_cstime : clock_t;
end
```

Record containing timings for fpTimes (178) call.

```
TnLink = UnixType.TnLink
```

Alias for nlink_t (119) type.

```
TOff = UnixType.TOff
```

Alias for off_t (119) type.

```
TPid = UnixType.TPid
```

Alias for pid_t (122) type.

```
TRLimit = record
  rlim_cur : rlim_t;
  rlim_max : rlim_t;
end
```

`TRLimit` is the structure used by the kernel to return resource limit information in.

```
tsigactionhandler = sigactionhandler_t
```

Alias for sigactionhandler_t (125) type.

```
tsigaltstack = record
  ss_sp : pointer;
  ss_flags : LongInt;
  ss_size : LongInt;
end
```

Provide the location of an alternate signal handler stack.

```
TSigContext = record
  gs : Word;
  __gsh : Word;
  fs : Word;
  __fsh : Word;
  es : Word;
  __esh : Word;
  ds : Word;
  __dsh : Word;
  edi : cardinal;
  esi : cardinal;
  ebp : cardinal;
  esp : cardinal;
  ebx : cardinal;
  edx : cardinal;
  ecx : cardinal;
  eax : cardinal;
  trapno : cardinal;
  err : cardinal;
  eip : cardinal;
  cs : Word;
  __csh : Word;
  eflags : cardinal;
  esp_at_signal : cardinal;
  ss : Word;
  __ssh : Word;
  fpstate : pfpstate;
  oldmask : cardinal;
  cr2 : cardinal;
end
```

This type is CPU dependent. Cross-platform code should not use the contents of this record.

```
tsiginfo = record
  si_signo : LongInt;
  si_errno : LongInt;
  si_code : LongInt;
  _sifields : record
  end;
end
```

This type describes the signal that occurred.

```
tsignalhandler = signalhandler_t
```

Alias for signalhandler_t (125) type.

```
tsigrestorerhandler = sigrestorerhandler_t
```

Alias for sigrestorerhandler_t (125) type.

```
tsigset = sigset_t
```

Alias for SigSet (126) type.

```
TSize = UnixType.TSize
```

Alias for size_t (126) type

```
TSocklen = UnixType.TSocklen
```

Alias for socklen_t (126) type.

```
TsSize = UnixType.TsSize
```

Alias for ssize_t (126) type

```
TStat = Stat
```

Alias for Stat (126) type.

```
tstatfs = UnixType.TStatFs
```

Record describing a file system in the baseunix.fpstatfs (94) call.

```
TTime = UnixType.TTime
```

Alias for TTime (130) type.

```
Ttimespec = UnixType.Ttimespec
```

Alias for TimeSpec (127) type.

```
TTimeVal = UnixType.TTimeVal
```

Alias for timeval (127) type.

```
TTimeZone = timezone
```

Alias for TimeZone (127) record.

```
TTms = tms
```

Alias for Tms (128) record type.

```
TUcontext = record
  uc_flags : cardinal;
  uc_link : Pucontext;
  uc_stack : tsigaltstack;
  uc_mcontext : TSigContext;
  uc_sigmask : tsigset;
end
```

This structure is used to describe the user context in a program or thread. It is not used in this unit, but is provided for completeness.

```
TUid = UnixType.TUid
```

Alias for uid_t (131) type.

```
TUtimBuf = UTimBuf
```

Alias for UtimBuf (131) type.

```
TUtsName = UtsName
```

Alias for UtsName (131) type.

```
uid_t = UnixType.uid_t
```

User ID type

```
UTimBuf = record
  actime : time_t;
  modtime : time_t;
end
```

Record used in fpUtime (180) to set file access and modificaton times.

```
UtsName = record
  Sysname : Array[0..UTSNAME_LENGTH-1] of Char;
  Nodename : Array[0..UTSNAME_NODENAME_LENGTH-1] of Char;
  Release : Array[0..UTSNAME_LENGTH-1] of Char;
  Version : Array[0..UTSNAME_LENGTH-1] of Char;
  Machine : Array[0..UTSNAME_LENGTH-1] of Char;
  Domain : Array[0..UTSNAME_DOMAIN_LENGTH-1] of Char;
end
```

The elements of this record are null-terminated C style strings, you cannot access them directly.

1.4 Procedures and functions

1.4.1 CreateShellArgV

Synopsis: Create a null-terminated array of strings from a command-line string

Declaration:
```
function CreateShellArgV(const prog: String) : ppchar
function CreateShellArgV(const prog: Ansistring) : ppchar
```

Visibility: default

Description: `CreateShellArgV` creates a command-line string for executing a shell command using 'sh -c'. The result is a null-terminated array of null-terminated strings suitable for use in fpExecv (138) and friends.

Errors: If no more memory is available, a heap error may occur.

See also: fpExecv (138), FreeShellArgV (182)

1.4.2 FpAccess

Synopsis: Check file access

Declaration:
```
function FpAccess(pathname: pChar;aMode: cint) : cint
function FpAccess(pathname: AnsiString;aMode: cint) : cint
```

Visibility: default

Description: `FpAccess` tests user's access rights on the specified file. `Mode` is a mask existing of one or more of the following:

R_OK User has read rights.

W_OK User has write rights.

X_OK User has execute rights.

F_OK File exists.

The test is done with the real user ID, instead of the effective user ID. If the user has the requested rights, zero is returned. If access is denied, or an error occurred, a nonzero value is returned.

Errors: Extended error information can be retrieved using fpGetErrno (145).

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_enoent A directory component in `Path` doesn't exist or is a dangling symbolic link.

sys_enotdir A directory component in `Path` is not a directory.

sys_enomem Insufficient kernel memory.

sys_eloop `Path` has a circular symbolic link.

See also: FpChown (135), FpChmod (134)

Listing: ./bunixex/ex26.pp

```
Program Example26;

{ Program to demonstrate the Access function. }

Uses BaseUnix;

begin
  if fpAccess ('/etc/passwd',W_OK)=0 then
    begin
```

```
    WriteIn ('Better check your system.');
    WriteIn ('I can write to the /etc/passwd file !');
    end;
end.
```

1.4.3 FpAlarm

Synopsis: Schedule an alarm signal to be delivered

Declaration: `function FpAlarm(seconds: cuint) : cuint`

Visibility: default

Description: `FpAlarm` schedules an alarm signal to be delivered to your process in `Seconds` seconds. When `Seconds` seconds have elapsed, the system will send a `SIGALRM` signal to the current process. If `Seconds` is zero, then no new alarm will be set. Whatever the value of `Seconds`, any previous alarm is cancelled.

The function returns the number of seconds till the previously scheduled alarm was due to be delivered, or zero if there was none. A negative value indicates an error.

See also: fpSigAction (168), fpPause (159)

Listing: ./bunixex/ex59.pp

```
Program Example59;

{ Program to demonstrate the Alarm function. }

Uses BaseUnix;

Procedure AlarmHandler(Sig : cint);cdecl;

begin
  WriteIn ('Got to alarm handler');
end;

begin
  WriteIn('Setting alarm handler');
  fpSignal(SIGALRM, SignalHandler(@AlarmHandler));
  WriteIn ('Scheduling Alarm in 10 seconds');
  fpAlarm(10);
  WriteIn ('Pausing');
  fpPause;
  WriteIn ('Pause returned');
end.
```

1.4.4 FpChdir

Synopsis: Change current working directory.

Declaration:
```
function FpChdir(path: pChar) : cint
function FpChdir(path: AnsiString) : cint
```

Visibility: default

Description: `fpChDir` sets the current working directory to `Path`.

It returns zero if the call was succesful, -1 on error.

Note: There exist a portable alterative to fpChDir: system.chdir. Please use fpChDir only if you are writing Unix specific code. System.chdir will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: fpGetCwd (144)

1.4.5 FpChmod

Synopsis: Change file permission bits

Declaration:
```
function FpChmod(path: pChar;Mode: TMode) : cint
function FpChmod(path: AnsiString;Mode: TMode) : cint
```

Visibility: default

Description: `fpChmod` sets the Mode bits of the file in `Path` to `Mode`. `Mode` can be specified by 'or'-ing the following values:

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.

If the function is successful, zero is returned. A nonzero return value indicates an error.

Errors: The following error codes are returned:

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_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: fpChown (135), fpAccess (132)

Listing: ./bunixex/ex23.pp

```
Program Example23;

{ Program to demonstrate the Chmod function. }

Uses BaseUnix,Unix;

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);
  fpChmod ('testex21',&777);
  { File is now executable  }
  fpexecl ('./testex21',[]);
end.
```

1.4.6 FpChown

Synopsis: Change owner of file

Declaration:
```
function FpChown(path: pChar;owner: TUid;group: TGid) : cint
function FpChown(path: AnsiString;owner: TUid;group: TGid) : cint
```

Visibility: default

Description: `fpChown` sets the User ID and Group ID of the file in `Path` to `Owner`,`Group`.

The function returns zero if the call was succesfull, a nonzero return value indicates an error.

Errors: The following error codes are returned:

sys_eperm The effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoent A directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

sys_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: fpChmod (134), fpAccess (132)

Listing: ./bunixex/ex24.pp

```
Program Example24;

{ Program to demonstrate the Chown function. }

Uses BaseUnix;

Var UID : TUid;
    GID : TGid;
    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 fpChown ('test.txt',UID,GID)<>0 then
    if fpgeterrno=ESysEPERM then
      Writeln ('You are not root !')
    else
      Writeln ('Chmod failed with exit code : ',fpgeterrno)
  else
    Writeln ('Changed owner successfully !');
end.
```

1.4.7 FpClose

Synopsis: Close file descriptor

Declaration: `function FpClose(fd: cint) : cint`

Visibility: default

Description: `FpClose` closes a file with file descriptor `Fd`. The function returns zero if the file was closed successfully, a nonzero return value indicates an error.

For an example, see FpOpen (157).

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpOpen (157), FpRead (161), FpWrite (182), FpFTruncate (144), FpLSeek (151)

1.4.8 FpClosedir

Synopsis: Close directory file descriptor

Declaration: `function FpClosedir(var dirp: Dir) : cint`

Visibility: default

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

For an example, see fpOpenDir (158).

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpOpenDir (158), FpReadDir (162)

1.4.9 FpDup

Synopsis: Duplicate a file handle

Declaration:
```
function FpDup(fildes: cint) : cint
function FpDup(var oldfile: text;var newfile: text) : cint
function FpDup(var oldfile: File;var newfile: File) : cint
```

Visibility: default

Description: `FpDup` returns a file descriptor that is a duplicate of the file descriptor `fildes`.

The second and third forms make `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile` in case it is a Text file or untyped file. Due to the buffering mechanism of Pascal, these calls do not have the same functionality as the `dup` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns a negative value in case of an error, a positive value is a file handle, and indicates succes.

Errors: A negative value can be one of the following error codes:

sys_ebadf `OldFile` hasn't been assigned.

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

See also: fpDup2 (137)

Listing: ./bunixex/ex31.pp

```
program Example31;

{ Program to demonstrate the Dup function. }

uses baseunix;

var f : text;

begin
  if fpdup(output,f)<>0 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.
```

1.4.10 FpDup2

Synopsis: Duplicate one filehandle to another

Declaration:
```
function FpDup2(fildes: cint;fildes2: cint) : cint
function FpDup2(var oldfile: text;var newfile: text) : cint
function FpDup2(var oldfile: File;var newfile: File) : cint
```

Visibility: default

Description: Makes `fildes2` or `NewFile` an exact copy of `fildes` or `OldFile`, after having flushed the buffer of `OldFile` in the case of text or untyped files.

After a call to fdup2, the 2 file descriptors point to the same physical device (a file, socket, or a terminal).

`NewFile` can be an assigned file. If `newfile` or `fildes` was open, it is closed first. Due to the buffering mechanism of Pascal, this has not the same functionality as the `dup2` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns zero if succesful, a nonzero return value means the call failed.

Errors: In case of error, the following error codes can be reported:

sys_ebadf`OldFile` (or `fildes`) hasn't been assigned.

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

See also: fpDup (137)

Listing: ./bunixex/ex32.pp

```
program Example31;

{ Program to demonstrate the FpDup2 function. }

uses BaseUnix;

var f : text;
    i : longint;

begin
  Assign (f,'text.txt');
  Rewrite (F);
  For i:=1 to 10 do writeln (F,'Line : ',i);
  if fpdup2 (output,f)=-1 then
    WriteIn ('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.}
  fpUnlink ('text.txt');
end.
```

1.4.11 FpExecv

Synopsis: Execute process

Declaration:
```
function FpExecv(path: pChar;argv: ppChar) : cint
function FpExecv(path: AnsiString;argv: ppchar) : cint
```

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argvp`. 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 fpErrno (94):

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

sys_epermThe file system is mounted \textit{noexec}.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: fpExecve (139), fpFork (142)

Listing: ./bunixex/ex8.pp

```
Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix, 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 }
  fpExecv ('/bin/ls',pp);
end.
```

1.4.12 FpExecve

Synopsis: Execute process using environment

Declaration:
```
function FpExecve(path: pChar;argv: ppChar;envp: ppChar) : cint
function FpExecve(path: AnsiString;argv: ppchar;envp: ppchar) : cint
```

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argv`, and the environment in `envp`. 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: Extended error information can be retrieved with fpGetErrno (145), and includes the following:

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

sys_epermThe file system is mounted \textit{noexec}.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: fpExecv (138), fpFork (142)

Listing: ./bunixex/ex7.pp

```
Program Example7;

{ Program to demonstrate the Execve function. }

Uses BaseUnix, 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 }
  fpExecVe ('/bin/ls',pp,envp);
end.
```

1.4.13 FpExit

Synopsis: Exit the current process

Declaration: `procedure FpExit(Status: cint)`

Visibility: default

Description: `FpExit` exits the currently running process, and report `Status` as the exit status.

Remark: If this call is executed, the normal unit finalization code will not be executed. This may lead to unexpected errors and stray files on your system. It is therefore recommended to use the `Halt` call instead.

Errors: None.

See also: FpFork (142), FpExecve (139)

1.4.14 FpFcntl

Synopsis: File control operations.

Declaration:
```
function FpFcntl(fildes: cint;cmd: cint) : cint
function FpFcntl(fildes: cint;cmd: cint;arg: cint) : cint
function FpFcntl(fildes: cint;cmd: cint;var arg: FLock) : cint
```

Visibility: default

Description: Read/set a file's attributes. `Fildes` a valid file descriptor. `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.

F_SetFd Set the close_on_exec flag of `fildes`. (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 the 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.

The function returns 0 if successful, -1 otherwise.

Errors: On error, -1 is returned. Use fpGetErrno (145) for extended error information.

sys_ebadf `Fd` has a bad file descriptor.

sys_eagain or sys_eaccess For \textbf{F_SetLk}, if the lock is held by another process.

1.4.15 fpfdfillset

Synopsis: Set all filedescriptors in the set.

Declaration:
```
function fpfdfillset(var nset: TFDSet) : cint
```

Visibility: default

Description: `fpfdfillset` sets all filedescriptors in `nset`.

See also: FpSelect (165), FpFD_ZERO (142), FpFD_IsSet (142), FpFD_Clr (141), FpFD_Set (142)

1.4.16 fpFD_CLR

Synopsis: Clears a filedescriptor in a set

Declaration:
```
function fpFD_CLR(fdno: cint;var nset: TFDSet) : cint
```

Visibility: default

Description: `FpFD_Clr` clears file descriptor `fdno` in filedescriptor set `nset`.

For an example, see FpSelect (165).

Errors: None.

See also: FpSelect (165), FpFD_ZERO (142), FpFD_Set (142), FpFD_IsSet (142)

1.4.17 fpFD_ISSET

Synopsis: Check whether a filedescriptor is set

Declaration: `function fpFD_ISSET(fdno: cint;const nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Set` Checks whether file descriptor `fdNo` in filedescriptor set `fds` is set. It returns zero if the descriptor is not set, 1 if it is set. If the number of the filedescriptor it wrong, -1 is returned.

For an example, see FpSelect (165).

Errors: If an invald file descriptor number is passed, -1 is returned.

See also: FpSelect (165), FpFD_ZERO (142), FpFD_Clr (141), FpFD_Set (142)

1.4.18 fpFD_SET

Synopsis: Set a filedescriptor in a set

Declaration: `function fpFD_SET(fdno: cint;var nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Set` sets file descriptor `fdno` in filedescriptor set `nset`.

For an example, see FpSelect (165).

Errors: None.

See also: FpSelect (165), FpFD_ZERO (142), FpFD_Clr (141), FpFD_IsSet (142)

1.4.19 fpFD_ZERO

Synopsis: Clear all file descriptors in set

Declaration: `function fpFD_ZERO(var nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_ZERO` clears all the filedescriptors in the file descriptor set `nset`.

For an example, see FpSelect (165).

Errors: None.

See also: FpSelect (165), FpFD_Clr (141), FpFD_Set (142), FpFD_IsSet (142)

1.4.20 FpFork

Synopsis: Create child process

Declaration: `function FpFork : TPid`

Visibility: default

Description: `FpFork` creates a child process which is a copy of the parent process. `FpFork` returns the process ID in the parent process, and zero in the child's process. (you can get the parent's PID with fpGetPPid (148)).

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: fpExecve (139), #rtl.linux.Clone (666)

1.4.21 FPFStat

Synopsis: Retrieve file information about a file descriptor.

Declaration:
```
function FpFStat(fd: cint;var sb: Stat) : cint
function FPFStat(var F: Text;var Info: Stat) : Boolean
function FPFStat(var F: File;var Info: Stat) : Boolean
```

Visibility: default

Description: `FpFStat` gets information about the file specified in one of the following:

Fda valid file descriptor.

Fan opened text file or untyped file.

and stores it in `Info`, which is of type stat (126). The function returns zero if the call was succesfull, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno (145).

sys_enoent`Path` does not exist.

See also: FpStat (173), FpLStat (152)

Listing: ./bunixex/ex28.pp

```
program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

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 fpstat ('test.fil',info)<>0 then
     begin
       writeln('Fstat failed. Errno : ',fpgeterrno);
       halt (1);
     end;
  writeln;
  writeln ('Result of fstat on file ''test.fil''.');
  writeln ('Inode    : ',info.st_ino);
  writeln ('Mode     : ',info.st_mode);
```

```
  writeln ('nlink   : ',info.st_nlink);
  writeln ('uid     : ',info.st_uid);
  writeln ('gid     : ',info.st_gid);
  writeln ('rdev    : ',info.st_rdev);
  writeln ('Size    : ',info.st_size);
  writeln ('Blksize : ',info.st_blksize);
  writeln ('Blocks  : ',info.st_blocks);
  writeln ('atime   : ',info.st_atime);
  writeln ('mtime   : ',info.st_mtime);
  writeln ('ctime   : ',info.st_ctime);
  { Remove file }
  erase (f);
end.
```

1.4.22 FpFtruncate

Synopsis: Truncate file on certain size.

Declaration: `function FpFtruncate(fd: cint;flength: TOff) : cint`

Visibility: default

Description: `FpFTruncate` sets the length of a file in `fd` on `flength` bytes, where `flength` must be less than or equal to the current length of the file in `fd`.

The function returns zero if the call was successful, a nonzero return value indicates that an error occurred.

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpOpen (157), FpClose (136), FpRead (161), FpWrite (182), FpLSeek (151)

1.4.23 FpGetcwd

Synopsis: Retrieve the current working directory.

Declaration: `function FpGetcwd(path: pChar;siz: TSize) : pChar`
`function FpGetcwd : AnsiString`

Visibility: default

Description: `fpgetCWD` returns the current working directory of the running process. It is returned in `Path`, which points to a memory location of at least `siz` bytes.

If the function is succesful, a pointer to `Path` is returned, or a string with the result. On error `Nil` or an empty string are returned.

Errors: On error `Nil` or an empty string are returned.

See also: FpGetPID (147), FpGetUID (148)

1.4.24 FpGetegid

Synopsis: Return effective group ID

Declaration: `function FpGetegid : TGid`

Visibility: default

Description: `FpGetegid` returns the effective group ID of the currently running process.

Errors: None.

See also: FpGetGid (146), FpGetUid (148), FpGetEUid (146), FpGetPid (147), FpGetPPid (148), fpSetUID (168), FpSetGid (167)

Listing: ./bunixex/ex18.pp

```
Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses BaseUnix;

begin
 writeln ('Group Id = ',fpgetgid,' Effective group Id = ',fpgetegid);
end.
```

1.4.25 FpGetEnv

Synopsis: Return value of environment variable.

Declaration:
```
function FpGetEnv(name: pChar) : pChar
function FpGetEnv(name: String) : pChar
```

Visibility: default

Description: `FPGetEnv` returns the value of the environment variable in `Name`. 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.

Listing: ./bunixex/ex41.pp

```
Program Example41;

{ Program to demonstrate the GetEnv function. }

Uses BaseUnix;

begin
  Writeln ('Path is : ',fpGetenv('PATH'));
end.
```

1.4.26 fpgeterrno

Synopsis: Retrieve extended error information.

Declaration:
```
function fpgeterrno : LongInt
```

Visibility: default

Description: `fpgeterrno` returns extended information on the latest error. It is set by all functions that communicate with the kernel or C library.

Errors: None.

See also: fpseterrno (166)

1.4.27 FpGeteuid

Synopsis: Return effective user ID

Declaration: `function FpGeteuid : TUid`

Visibility: default

Description: `FpGeteuid` returns the effective user ID of the currently running process.

Errors: None.

See also: FpGetUid (148), FpGetGid (146), FpGetEGid (144), FpGetPid (147), FpGetPPid (148), fpSetUID (168), FpSetGid (167)

Listing: ./bunixex/ex17.pp

```
Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses BaseUnix;

begin
  writeln ('User Id = ',fpgetuid,' Effective user Id = ',fpgeteuid);
end.
```

1.4.28 FpGetgid

Synopsis: Return real group ID

Declaration: `function FpGetgid : TGid`

Visibility: default

Description: `FpGetgid` returns the real group ID of the currently running process.

Errors: None.

See also: FpGetEGid (144), FpGetUid (148), FpGetEUid (146), FpGetPid (147), FpGetPPid (148), fpSetUID (168), FpSetGid (167)

Listing: ./bunixex/ex18.pp

```
Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses BaseUnix;

begin
 writeln ('Group Id = ',fpgetgid,' Effective group Id = ',fpgetegid);
end.
```

1.4.29 FpGetgroups

Synopsis: Get the list of supplementary groups.

Declaration: `function FpGetgroups(gidsetsize: cint;var grouplist: TGrpArr) : cint`

Visibility: default

Description: `FpGetgroups` returns up to `gidsetsize` groups in `GroupList`

If the function is successful, then number of groups that were stored is returned. On error, -1 is returned.

Errors: On error, -1 is returned. Extended error information can be retrieved with fpGetErrNo (145)

See also: FpGetpgrp (147), FpGetGID (146), FpGetEGID (144)

1.4.30 FpGetpgrp

Synopsis: Get process group ID

Declaration: `function FpGetpgrp : TPid`

Visibility: default

Description: `FpGetpgrp` returns the process group ID of the current process.

Errors: None.

See also: fpGetPID (147), fpGetPPID (148), FpGetGID (146), FpGetUID (148)

1.4.31 FpGetpid

Synopsis: Return current process ID

Declaration: `function FpGetpid : TPid`

Visibility: default

Description: `FpGetpid` returns the process ID of the currently running process.

Note: There exist a portable alterative to fpGetpid: system.GetProcessID. Please use fpGetpid only if you are writing Unix specific code. System.GetProcessID will work on all operating systems.

Errors: None.

See also: FpGetPPid (148)

Listing: ./bunixex/ex16.pp

```
Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses BaseUnix;

begin
  Writeln ('Process Id = ',fpgetpid,' Parent process Id = ',fpgetppid);
end.
```

1.4.32 FpGetppid

Synopsis: Return parent process ID

Declaration: `function FpGetppid : TPid`

Visibility: default

Description: `FpGetppid` returns the Process ID of the parent process.

Errors: None.

See also: FpGetPid (147)

Listing: ./bunixex/ex16.pp

```
Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses BaseUnix;

begin
  WriteIn ('Process Id = ',fpgetpid,' Parent process Id = ',fpgetppid);
end.
```

1.4.33 fpGetPriority

Synopsis: Return process priority

Declaration: `function fpGetPriority(Which: cint;Who: cint) : cint`

Visibility: default

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.

For an example, see FpNice (156).

Errors: Error information is returned solely by the FpGetErrno (145) function: a priority can be a positive or negative value.

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: FpSetPriority (167), FpNice (156)

1.4.34 FpGetuid

Synopsis: Return current user ID

Declaration: `function FpGetuid : TUid`

Visibility: default

Description: `FpGetuid` returns the real user ID of the currently running process.

Errors: None.

See also: FpGetGid (146), FpGetEUid (146), FpGetEGid (144), FpGetPid (147), FpGetPPid (148), fpSetUID (168)

Listing: ./bunixex/ex17.pp

```
Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses BaseUnix;

begin
  writeln ('User Id = ',fpgetuid,' Effective user Id = ',fpgeteuid);
end.
```

1.4.35 FpIOCtl

Synopsis: General kernel IOCTL call.

Declaration: `function FpIOCtl(Handle: cint;Ndx: TIOCtlRequest;Data: Pointer) : cint`

Visibility: default

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: Extended error information can be retrieved using fpGetErrno (145).

Listing: ./bunixex/ex54.pp

```
Program Example54;

uses BaseUnix,Termio;

{ Program to demonstrate the IOCtl function. }

var
  tios : Termios;

begin
  {$ifdef FreeBSD}
    fpIOCtl(1,TIOCGETA,@tios);   // these constants are very OS dependant.
                                 // see the tcgetattr example for a better way
  {$endif}
  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.
```

1.4.36 FpKill

Synopsis: Send a signal to a process

Declaration: `function FpKill(pid: TPid;sig: cint) : cint`

Visibility: default

Description: `fpKill` sends 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: Extended error information can be retrieved using fpGetErrno (145):

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: FpSigAction (168), FpSignal (171)

1.4.37 FpLink

Synopsis: Create a hard link to a file

Declaration:
```
function FpLink(existing: pChar;newone: pChar) : cint
function FpLink(existing: AnsiString;newone: AnsiString) : cint
```

Visibility: default

Description: `fpLink` makes `NewOne` point to the same file als `Existing`. The two files then have the same inode number. This is known as a 'hard' link. The function returns `zero` if the call was succesfull, and returns a `non-zero` value if the call failed.

Errors: The following error codes are returned:

sys_exdev`Existing` and `NewOne` are not on the same filesystem.

sys_eperm The filesystem containing `Existing` and `NewOne` doesn't support linking files.

sys_eaccess Write access for the directory containing `NewOne` is disallowed, or one of the directories in `Existing` or `NewOne` has no search (=execute) permission.

sys_enoentA directory entry in `Existing` or `NewOne` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdirA directory entry in `Existing` or `NewOne` is nor a directory.

sys_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only filesystem.

sys_eexist`NewOne` already exists.

sys_emlink`Existing` has reached maximal link count.

sys_eloop`existing` or `NewOne` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospc The device containing `NewOne` has no room for another entry.

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

See also: fpSymLink (174), fpUnLink (179)

Listing: ./bunixex/ex21.pp

```
Program Example21;

{ Program to demonstrate the Link and UnLink functions. }

Uses BaseUnix;

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 fpLink ('test.txt','new.txt')<>0 then
    writeln ('Error when linking !');
  { Removing test.txt still leaves new.txt }
  If fpUnlink ('test.txt')<>0 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 FPUnlink ('new.txt')<>0 then
   Writeln ('Error when unlinking !');
end.
```

1.4.38 FpLseek

Synopsis: Set file pointer position.

Declaration: `function FpLseek(fd: cint;offset: TOff;whence: cint) : TOff`

Visibility: default

Description: `FpLSeek` sets the current fileposition of file `fd` to `Offset`, starting from `Whence`, which can be one of the following:

Seek_Set`Offset` is the absolute position in the file.

Seek_Cur`Offset` is relative to the current position.

Seek_end`Offset` is relative to the end of the file.

The function returns the new fileposition, or -1 of an error occurred.

For an example, see FpOpen (157).

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpOpen (157), FpWrite (182), FpClose (136), FpRead (161), FpFTruncate (144)

1.4.39 fpLstat

Synopsis: Return information about symbolic link. Do not follow the link

Declaration:
```
function fpLstat(path: pchar;Info: PStat) : cint
function fpLstat(Filename: ansistring;Info: PStat) : cint
```

Visibility: default

Description: `FpLstat` gets information about the link specified in `Path` (or `FileName`, and stores it in `Info`, which points to a record of type `TStat`. Contrary to FpFstat (143), it stores information about the link, not about the file the link points to. The function returns zero if the call was succesful, a nonzero return value indicates failure. failed.

Errors: Extended error information is returned by the FpGetErrno (145) function.

sys_enoent`Path` does not exist.

See also: FpFStat (143), #rtl.unix.StatFS (1572)

Listing: ./unixex/ex29.pp

```
program example29;

{ Program to demonstrate the LStat function. }

uses BaseUnix,Unix;

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 fpstat ('test.fil',info)<>0 then
     begin
     writeln('Fstat failed. Errno : ',fpgeterrno);
     halt (1);
     end;
  writeln;
  writeln ('Result of stat on file ''test.fil''.');
  writeln ('Inode   : ',info.st_ino);
  writeln ('Mode    : ',info.st_mode);
  writeln ('nlink   : ',info.st_nlink);
  writeln ('uid     : ',info.st_uid);
  writeln ('gid     : ',info.st_gid);
  writeln ('rdev    : ',info.st_rdev);
  writeln ('Size    : ',info.st_size);
  writeln ('Blksize : ',info.st_blksize);
  writeln ('Blocks  : ',info.st_blocks);
  writeln ('atime   : ',info.st_atime);
  writeln ('mtime   : ',info.st_mtime);
  writeln ('ctime   : ',info.st_ctime);

  If  fpSymLink ('test.fil','test.lnk')<>0 then
```

```
    writeln ('Link failed ! Errno :',fpgeterrno);

  if  fplstat ('test.lnk',@info)<>0 then
     begin
     writeln('LStat failed. Errno : ',fpgeterrno);
     halt (1);
     end;
  writeln;
  writeln ('Result of fstat on file ''test.lnk''.');
  writeln ('Inode    : ',info.st_ino);
  writeln ('Mode     : ',info.st_mode);
  writeln ('nlink    : ',info.st_nlink);
  writeln ('uid      : ',info.st_uid);
  writeln ('gid      : ',info.st_gid);
  writeln ('rdev     : ',info.st_rdev);
  writeln ('Size     : ',info.st_size);
  writeln ('Blksize  : ',info.st_blksize);
  writeln ('Blocks   : ',info.st_blocks);
  writeln ('atime    : ',info.st_atime);
  writeln ('mtime    : ',info.st_mtime);
  writeln ('ctime    : ',info.st_ctime);
  { Remove file and link }
  erase (f);
  fpunlink ('test.lnk');
end.
```

1.4.40 FpMkdir

Synopsis: Create a new directory

Declaration:
```
function FpMkdir(path: pChar;Mode: TMode) : cint
function FpMkdir(path: AnsiString;Mode: TMode) : cint
```

Visibility: default

Description: `FpMkDir` creates a new directory `Path`, and sets the new directory's mode to `Mode`. Path can be an absolute path or a relative path. Note that only the last element of the directory will be created, higher level directories must already exist, and must be writeable by the current user.

On succes, 0 is returned. if the function fails, -1 is returned.

Note: There exist a portable alterative to fpMkDir: system.mkdir. Please use fpMkDir only if you are writing Unix specific code. System.mkdir will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: fpGetCWD (144), fpChDir (133)

1.4.41 FpMkfifo

Synopsis: Create FIFO (named pipe) in file system

Declaration:
```
function FpMkfifo(path: pChar;Mode: TMode) : cint
function FpMkfifo(path: AnsiString;Mode: TMode) : cint
```

Visibility: default

Description: `fpMkFifo` creates named a named pipe in the filesystem, with name `Path` and mode `Mode`.

The function returns zero if the command was succesful, and nonzero if it failed.

Errors: The error codes include:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

1.4.42 Fpmmap

Synopsis: Create memory map of a file

Declaration:
```
function Fpmmap(start: pointer;len: size_t;prot: cint;flags: cint;
                fd: cint;offst: off_t) : pointer
```

Visibility: default

Description: `FpMMap` maps or unmaps files or devices into memory. The different arguments determine what and how the file is mapped:

adr Address where to mmap the device. This address is a hint, and may not be followed.

len Size (in bytes) of area to be mapped.

prot Protection of mapped memory. This is a OR-ed combination of the following constants:

- **PROT_EXEC** The memory can be executed.
- **PROT_READ** The memory can be read.
- **PROT_WRITE** The memory can be written.
- **PROT_NONE** The memory can not be accessed.

flags Contains some options for the mmap call. It is an OR-ed combination of the following constants:

- **MAP_FIXED** Do not map at another address than the given address. If the address cannot be used, `MMap` will fail.
- **MAP_SHARED** Share this map with other processes that map this object.
- **MAP_PRIVATE** Create a private map with copy-on-write semantics.
- **MAP_ANONYMOUS** `fd` does not have to be a file descriptor.

One of the options `MAP_SHARED` and `MAP_PRIVATE` must be present, but not both at the same time.

fd File descriptor from which to map.

off Offset to be used in file descriptor `fd`.

The function returns a pointer to the mapped memory, or a -1 in case of en error.

Errors: On error, -1 is returned and extended error information is returned by the FpGetErrno (145) function.

Sys_EBADF `fd` is not a valid file descriptor and `MAP_ANONYMOUS` was not specified.

Sys_EACCES `MAP_PRIVATE` was specified, but `fd` is not open for reading. Or `MAP_SHARED` was asked and `PROT_WRITE` is set, fd is not open for writing

Sys_EINVAL One of the record fields `Start`, `length` or `offset` is invalid.

Sys_ETXTBUSY `MAP_DENYWRITE` was set but the object specified by fd is open for writing.

Sys_EAGAIN `fd` is locked, or too much memory is locked.

Sys_ENOMEM Not enough memory for this operation.

See also: FpMUnMap (155)

Listing: ./unixex/ex66.pp

```
Program Example66;

{ Program to demonstrate the MMap function. }

Uses BaseUnix,Unix;

Var S     : String;
    fd    : cint;
    Len   : longint;
//    args : tmmapargs;
    P     : PChar;

begin
  s:='This is the string';
  Len:=Length(S);
  fd:=fpOpen('testfile.txt',O_wrOnly or o_creat);
  If fd=-1 then
    Halt(1);
  If fpWrite(fd,S[1],Len)=-1 then
    Halt(2);
  fpClose(fd);
  fd:=fpOpen('testfile.txt',O_rdOnly);
  if fd=-1 then
    Halt(3);
  P:=Pchar(fpmmap(nil,len+1 ,PROT_READ or PROT_WRITE,MAP_PRIVATE,fd,0));

  If longint(P)=-1 then
    Halt(4);
  Writeln('Read in memory  :',P);
  fpclose(fd);
  if fpMUnMap(P,Len)<>0 Then
    Halt(fpgeterrno);
end.
```

1.4.43 Fpmunmap

Synopsis: Unmap previously mapped memory block

Declaration: `function Fpmunmap(start: pointer;len: size_t) : cint`

Visibility: default

Description: `FpMUnMap` unmaps the memory block of size `Len`, pointed to by `Adr`, which was previously allocated with FpMMap (154).

The function returns `True` if successful, `False` otherwise.

For an example, see FpMMap (154).

Errors: In case of error the function returns a nonzero value, extended error information is returned by the FpGetErrno (145) function. See FpMMap (154) for possible error values.

See also: FpMMap (154)

1.4.44 FpNanoSleep

Synopsis: Suspend process for a short time

Declaration: `function FpNanoSleep(req: ptimespec;rem: ptimespec) : cint`

Visibility: default

Description: `FpNanoSleep` suspends the process till a time period as specified in `req` has passed. Then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and `rem` will contain the remaining time till the end of the intended period. In this case the return value will be -1, and `fpErrNo` will be set to `EINTR`

If the function returns without error, the return value is zero.

Errors: If an error occurred or the call was interrupted, -1 is returned. Extended error information can be retrieved using fpGetErrno (145).

See also: FpPause (159), FpAlarm (133)

Listing: ./bunixex/ex72.pp

```
program example72;

{ Program to demonstrate the NanoSleep function. }

uses BaseUnix;

Var
  Req,Rem : TimeSpec;
  Res : Longint;

begin
  With Req do
    begin
    tv_sec:=10;
    tv_nsec:=100;
    end;
  Write('NanoSleep returned : ');
  Flush(Output);
  Res:=(fpNanoSleep(@Req,@rem));
  Writeln(res);
  If (res<>0) then
    With rem do
      begin
      Writeln('Remaining seconds      : ',tv_sec);
      Writeln('Remaining nanoseconds : ',tv_nsec);
      end;
end.
```

1.4.45 fpNice

Synopsis: Set process priority

Declaration: `function fpNice(N: cint) : cint`

Visibility: default

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.

If the function is succesful, zero is returned. On error, a nonzero value is returned.

Errors: Extended error information is returned by the FpGetErrno (145) function.

sys_epermA non-superuser tried to specify a negative `N`, i.e. do a priority increase.

See also: FpGetPriority (148), FpSetPriority (167)

Listing: ./unixex/ex15.pp

```
Program Example15;

{ Program to demonstrate the Nice and Get/SetPriority functions. }

Uses BaseUnix, Unix;

begin
  writeln ('Setting priority to 5');
  fpsetpriority (prio_process,fpgetpid,5);
  writeln ('New priority = ',fpgetpriority (prio_process,fpgetpid));
  writeln ('Doing nice 10');
  fpnice (10);
  writeln ('New Priority = ',fpgetpriority (prio_process,fpgetpid));
end.
```

1.4.46 FpOpen

Synopsis: Open file and return file descriptor

Declaration:
```
function FpOpen(path: pChar;flags: cint;Mode: TMode) : cint
function FpOpen(path: pChar;flags: cint) : cint
function FpOpen(path: AnsiString;flags: cint) : cint
function FpOpen(path: AnsiString;flags: cint;Mode: TMode) : cint
function FpOpen(path: String;flags: cint) : cint
function FpOpen(path: String;flags: cint;Mode: TMode) : cint
```

Visibility: default

Description: `FpOpen` opens a file in `Path` with flags `flags` and mode `Mode` One of the following:

O_RdOnly File is opened Read-only

O_WrOnly File is opened Write-only

O_RdWr File is opened Read-Write

The flags may be`OR`-ed with one of the following constants:

O_Creat File is created if it doesn't exist.

O_Excl If the file is opened with `O_Creat` and it already exists, the call wil fail.

O_NoCtty If the file is a terminal device, it will NOT become the process' controlling terminal.

O_Trunc If the file exists, it will be truncated.

O_Appendthe file is opened in append mode. *Before each write*, the file pointer is positioned at the end of the file.

O_NonBlock The file is opened in non-blocking mode. No operation on the file descriptor will cause the calling process to wait till.

O_NDelay Idem as `O_NonBlock`

O_Sync The file is opened for synchronous IO. Any write operation on the file will not return untill the data is physically written to disk.

O_NoFollowif the file is a symbolic link, the open fails. (linux 2.1.126 and higher only)

O_Directoryif the file is not a directory, the open fails. (linux 2.1.126 and higher only)

`Path` 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: Extended error information can be retrieved using fpGetErrno (145).

See also: FpClose (136), FpRead (161), FpWrite (182), FpFTruncate (144), FpLSeek (151)

Listing: ./bunixex/ex19.pp

```
Program Example19;

{ Program to demonstrate the fdOpen, fdwrite and fdCLose functions. }

Uses BaseUnix;

Const Line : String[80] = 'This is easy writing !';

Var FD : Cint;

begin
  FD:=fpOpen ('Test.dat',O_WrOnly or O_Creat);
  if FD>0 then
    begin
    if length(Line)<>fpwrite (FD,Line[1],Length(Line)) then
      WriteLn ('Error when writing to file !');
    fpClose(FD);
    end;
end.
```

1.4.47 FpOpendir

Synopsis: Open a directory for reading

Declaration:
```
function FpOpendir(dirname: pChar) : pDir
function FpOpendir(dirname: AnsiString) : pDir
function FpOpendir(dirname: shortString) : pDir
```

Visibility: default

Description: `FpOpenDir` opens the directory `DirName`, and returns a `pdir` pointer to a Dir (118) record, which can be used to read the directory structure. If the directory cannot be opened, `nil` is returned.

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpCloseDir (136), FpReadDir (162)

Listing: ./bunixex/ex35.pp

```
Program Example35;

{ Program to demonstrate the
  OpenDir,ReadDir, SeekDir and TellDir functions. }

Uses BaseUnix;

Var TheDir : PDir;
    ADirent : PDirent;
    Entry : Longint;

begin
  TheDir:=fpOpenDir('./.');
  Repeat
//    Entry:=fpTellDir(TheDir);
    ADirent:=fpReadDir (TheDir^);
    If ADirent<>Nil then
      With ADirent^ do
        begin
        Writeln ('Entry No : ',Entry);
        Writeln ('Inode    : ',d_fileno);
//        Writeln ('Offset   : ',d_off);
        Writeln ('Reclen   : ',d_reclen);
        Writeln ('Name     : ',pchar(@d_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
//      fpSeekDir (TheDir,Entry);                // not implemented for various platforms
      ADirent:=fpReadDir (TheDir^);
      If ADirent<>Nil then
        With ADirent^ do
          begin
          Writeln ('Entry No : ',Entry);
          Writeln ('Inode    : ',d_fileno);
//          Writeln ('Offset   : ',off);
          Writeln ('Reclen   : ',d_reclen);
          Writeln ('Name     : ',pchar(@d_name[0]));
          end;
    end;
  Until Entry=-1;
  fpCloseDir (TheDir^);
end.
```

1.4.48 FpPause

Synopsis: Wait for a signal to arrive

Declaration: `function FpPause : cint`

Visibility: default

Description: `FpPause` puts the process to sleep and waits until the application receives a signal. If a signal handler is installed for the received sigal, the handler will be called and after that pause will return control to the process.

For an example, see fpAlarm (133).

1.4.49 FpPipe

Synopsis: Create a set of pipe file handlers

Declaration: `function FpPipe(var fildes: TFilDes) : cint`

Visibility: default

Description: `FpPipe` creates a pipe, i.e. two file objects, one for input, one for output. The filehandles are returned in the array `fildes`. The input handle is in the 0-th element of the array, the output handle is in the 1-st element.

The function returns zero if everything went succesfully, a nonzero return value indicates an error.

Errors: In case the function fails, the following return values are possible:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: #rtl.unix.POpen (1568), fpMkFifo (153)

Listing: ./bunixex/ex36.pp

```
Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix,Unix;

Var pipi,pipo : Text;
    s : String;

begin
  Writeln ('Assigning Pipes.');
  If assignpipe(pipi,pipo)<>0 then
    Writeln('Error assigning pipes !',fpgeterrno);
  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.
```

1.4.50 FppRead

Synopsis: Positional read: read from file descriptor at a certain position.

Declaration:
```
function FpPRead(fd: cint;buf: pChar;nbytes: TSize;offset: TOff)
                    : TsSize
function FppRead(fd: cint;var buf;nbytes: TSize;offset: TOff) : TsSize
```

Visibility: default

Description: `FpPRead` reads `nbytes` bytes from file descriptor `fd` into buffer `buf` starting at offset `offset`. `Offset` is measured from the start of the file. This function can only be used on files, not on pipes or sockets (i.e. any seekable file descriptor).

The function returns the number of bytes actually read, or -1 on error.

Errors: On error, -1 is returned.

See also: FpReadV (164), FpPWrite (161)

1.4.51 FppWrite

Synopsis: Positional write: write to file descriptor at a certain position.

Declaration:
```
function FpPWrite(fd: cint;buf: pChar;nbytes: TSize;offset: TOff)
                    : TsSize
function FppWrite(fd: cint;const buf;nbytes: TSize;offset: TOff)
                    : TsSize
```

Visibility: default

Description: `FpPWrite` writes `nbytes` bytes from buffer `buf` into file descriptor `fd` starting at offset `offset`. `Offset` is measured from the start of the file. This function can only be used on files, not on pipes or sockets (i.e. any seekable file descriptor).

The function returns the number of bytes actually written, or -1 on error.

Errors: On error, -1 is returned.

See also: FpPRead (161), FpWriteV (182)

1.4.52 FpRead

Synopsis: Read data from file descriptor

Declaration:
```
function FpRead(fd: cint;buf: pChar;nbytes: TSize) : TsSize
function FpRead(fd: cint;var buf;nbytes: TSize) : TsSize
```

Visibility: default

Description: `FpdRead` reads at most `nbytes` 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: Extended error information can be retrieved using fpGetErrno (145).

See also: FpOpen (157), FpClose (136), FpWrite (182), FpFTruncate (144), FpLSeek (151)

Listing: ./bunixex/ex20.pp

```
Program Example20;

{ Program to demonstrate the fdRead and fdTruncate functions. }

Uses BaseUnix;

Const Data : string[10] = '1234567890';

Var FD : cint;
    l : longint;

begin
  FD:=fpOpen('test.dat',o_wronly or o_creat,&666);
  if fd>0 then
    begin
    { Fill file with data }
    for l:=1 to 10 do
      if fpWrite (FD,Data[1],10)<>10 then
        begin
        writeln ('Error when writing !');
        halt(1);
        end;
    fpClose(FD);
    FD:=fpOpen('test.dat',o_rdonly);
    { Read data again }
    If FD>0 then
      begin
      For l:=1 to 5 do
        if fpRead (FD,Data[1],10)<>10 then
          begin
          WriteIn ('Error when Reading !');
          Halt(2);
          end;
      fpClose(FD);
      { Truncating file at 60 bytes }
      { For truncating, file must be open or write }
      FD:=fpOpen('test.dat',o_wronly,&666);
      if FD>0 then
        begin
        if fpfTruncate(FD,60)<>0 then
           WriteIn('Error when truncating !');
        fpClose (FD);
        end;
      end;
    end;
end.
```

1.4.53 FpReaddir

Synopsis: Read entry from directory

Declaration: `function FpReaddir(var dirp: Dir) : pDirent`

Visibility: default

Description: `FpReadDir` reads the next entry in the directory pointed to by `dirp`. It returns a `pdirent` pointer to a dirent (119) record describing the entry. If the next entry can't be read, `Nil` is returned.

For an example, see FpOpenDir (158).

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpCloseDir (136), FpOpenDir (158)

1.4.54 fpReadLink

Synopsis: Read destination of symbolic link

Declaration:
```
function fpReadLink(name: pchar;linkname: pchar;maxlen: size_t) : cint
function fpReadLink(Name: ansistring) : ansistring
```

Visibility: default

Description: `FpReadLink` returns the file the symbolic link `name` is pointing to. The first form of this function accepts a buffer `linkname` of length `maxlen` where the filename will be stored. It returns the actual number of characters stored in the buffer.

The second form of the function returns simply the name of the file.

Errors: On error, the first form of the function returns -1; the second one returns an empty string. Extended error information is returned by the FpGetErrno (145) function.

SYS_ENOTDIR A part of the path in `Name` is not a directory.

SYS_EINVAL maxlen is not positive, or the file is not a symbolic link.

SYS_ENAMETOOLONG A pathname, or a component of a pathname, was too long.

SYS_ENOENT the link `name` does not exist.

SYS_EACCES No permission to search a directory in the path

SYS_ELOOP Too many symbolic links were encountered in translating the pathname.

SYS_EIO An I/O error occurred while reading from the file system.

SYS_EFAULT The buffer is not part of the the process's memory space.

SYS_ENOMEM Not enough kernel memory was available.

See also: FpSymLink (174)

Listing: ./unixex/ex62.pp

```
Program Example62;

{ Program to demonstrate the ReadLink function. }

Uses BaseUnix,Unix;

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 fpSymLink ('test.txt','new.txt')<>0 then
    writeln ('Error when symlinking !');
  S:=fpReadLink('new.txt');
```

```
  If S='' then
    WriteIn ('Error reading link !')
  Else
    WriteIn ('Link points to : ',S);
 { Now remove links }
 If fpUnlink ('new.txt')<>0 then
   WriteIn ('Error when unlinking !');
 If fpUnlink ('test.txt')<>0 then
   WriteIn ('Error when unlinking !');
end.
```

1.4.55 FpReadV

Synopsis: Vector read: Read into multiple buffers

Declaration: `function FpReadV(fd: cint;const iov: piovec;iovcnt: cint) : TsSize`

Visibility: default

Description: `FpReadV` reads data from file descriptor `fd` and writes it into `iovcnt` buffers described by the tiovec (128) buffers pointed to by `iov`. It works like fpRead (161) only on multiple buffers.

Errors: On error, -1 is returned.

See also: FpWriteV (182), FpPWrite (161), FpPRead (161)

1.4.56 FpRename

Synopsis: Rename file

Declaration:
```
function FpRename(old: pChar;newpath: pChar) : cint
function FpRename(old: AnsiString;newpath: AnsiString) : cint
```

Visibility: default

Description: `FpRename` renames the file `Old` to `NewPath`. `NewPath` can be in a different directory than `Old`, but it cannot be on another partition (device). Any existing file on the new location will be replaced.

If the operation fails, then the `Old` file will be preserved.

The function returns zero on succes, a nonzero value indicates failure.

Note: There exist a portable alterative to fpRename: system.rename. Please use fpRename only if you are writing Unix specific code. System.rename will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (145).

sys_eisdir`NewPath` exists and is a directory, but `Old` is not a directory.

sys_exdev`NewPath` and `Old` are on different devices.

sys_enotempty or sys_eexist`NewPath` is an existing, non-empty directory.

sys_ebusy`Old` or `NewPath` is a directory and is in use by another process.

sys_einval`NewPath` is part of `Old`.

sys_emlink`OldPath` or `NewPath` already have tha maximum amount of links pointing to them.

sys_enotdirpart of `Old` or `NewPath` is not directory.

sys_efault For the `pchar` case: One of the pointers points to an invalid address.

sys_eaccessaccess is denied when attempting to move the file.

sys_enametoolong Either `Old` or `NewPath` is too long.

sys_enoenta directory component in `Old` or `NewPath` didn't exist.

sys_enomemnot enough kernel memory.

sys_erofs`NewPath` or `Old` is on a read-only file system.

sys_elooptoo many symbolic links were encountered trying to expand `Old` or `NewPath`

sys_enospcthe filesystem has no room for the new directory entry.

See also: FpUnLink (179)

1.4.57 FpRmdir

Synopsis: Remove a directory.

Declaration:
```
function FpRmdir(path: pChar) : cint
function FpRmdir(path: AnsiString) : cint
```

Visibility: default

Description: `FpRmdir` removes the directory `Path` from the system. The directory must be empty for this call to succeed, and the user must have the necessary permissions in the parent directory. Only the last component of the directory is removed, i.e. higher-lying directories are not removed.

On success, zero is returned. A nonzero return value indicates failure.

Note: There exist a portable alterative to fpRmDir: system.rmdir. Please use fpRmDir only if you are writing Unix specific code. System.rmdir will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (145).

1.4.58 fpSelect

Synopsis: Wait for events on file descriptors

Declaration:
```
function FPSelect(N: cint;readfds: pFDSet;writefds: pFDSet;
                  exceptfds: pFDSet;TimeOut: ptimeval) : cint
function fpSelect(N: cint;readfds: pFDSet;writefds: pFDSet;
                  exceptfds: pFDSet;TimeOut: cint) : cint
function fpSelect(var T: Text;TimeOut: ptimeval) : cint
function fpSelect(var T: Text;TimeOut: time_t) : cint
```

Visibility: default

Description: `FpSelect` 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 fpFD_ZERO (142), fpFD_Clr (141), fpFD_Set (142) or fpFD_IsSet (142) to manipulate the individual elements of a set.

The pointers can be `Nil`.

N is the value of the largest file descriptor in one of the sets, + 1. In other words, it is the position of the last bit which is set in the array of bits.

`TimeOut` can be used to set a time limit. If `TimeOut` can be two types :

1. `TimeOut` is of type `ptimeval` 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 `cint`. 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.

The variant with the text file will execute the `FpSelect` call on the file descriptor associated with the text file `T`

Errors: On error, the function returns -1. Extended error information can be retrieved using fpGetErrno (145).

SYS_EBADF An invalid descriptor was specified in one of the sets.

SYS_EINTR A non blocked signal was caught.

SYS_EINVAL `N` is negative or too big.

SYS_ENOMEM `Select` was unable to allocate memory for its internal tables.

See also: fpFD_ZERO (142), fpFD_Clr (141), fpFD_Set (142), fpFD_IsSet (142)

Listing: ./bunixex/ex33.pp

```
Program Example33;

{ Program to demonstrate the Select function. }

Uses BaseUnix;

Var FDS : Tfdset;

begin
  fpfd_zero(FDS);
  fpfd_set(0,FDS);
  Writeln ('Press the <ENTER> to continue the program.');
  { Wait until File descriptor 0 (=Input) changes }
  fpSelect (1,@FDS,nil,nil,nil);
  { Get rid of <ENTER> in buffer }
  readln;
  Writeln ('Press <ENTER> key in less than 2 seconds...');
  Fpfd_zero(FDS);
  FpFd_set (0,FDS);
  if fpSelect (1,@FDS,nil,nil,2000)>0 then
    Writeln ('Thank you !')
    { FD_ISSET(0,FDS) would be true here. }
  else
    Writeln ('Too late !');
end.
```

1.4.59 fpseterrno

Synopsis: Set extended error information.

Declaration: `procedure fpseterrno(err: LongInt)`

Visibility: default

Description: `fpseterrno` sets the extended information on the latest error. It is called by all functions that communicate with the kernel or C library.

Unless a direct kernel call is performed, there should never be any need to call this function.

Errors:

See also: fpgeterrno (145)

1.4.60 FpSetgid

Synopsis: Set the current group ID

Declaration: `function FpSetgid(gid: TGid) : cint`

Visibility: default

Description: `fpSetUID` sets the group ID of the current process. This call will only work if it is executed as root, or the program is setgid root.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with fpGetErrNo (145).

See also: FpSetUid (168), FpGetGid (146), FpGetUid (148), FpGetEUid (146), FpGetEGid (144), FpGetPid (147), FpGetPPid (148)

1.4.61 fpSetPriority

Synopsis: Set process priority

Declaration: `function fpSetPriority(Which: cint;Who: cint;What: cint) : cint`

Visibility: default

Description: `fpSetPriority` 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 constants:

Prio_Process`Who` is interpreted as process ID

Prio_PGrp`Who` is interpreted as process group ID

Prio_User`Who` is interpreted as user ID

`Prio` is a value in the range -20 to 20.

For an example, see FpNice (156).

The function returns zero on success, -1 on failure

Errors: Extended error information is returned by the FpGetErrno (145) function.

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: FpGetPriority (148), FpNice (156)

1.4.62 FpSetsid

Synopsis: Create a new session.

Declaration: `function FpSetsid : TPid`

Visibility: default

Description: `FpSetsid` creates a new session (process group). It returns the new process group id (as returned by FpGetpgrp (147). This call will fail if the current process is already the process group leader.

Errors: On error, -1 is returned. Extended error information can be retrieved with fpGetErrNo (145)

1.4.63 fpsettimeofday

Synopsis: Set kernel time

Declaration: `function fpsettimeofday(tp: ptimeval;tzp: ptimezone) : cint`

Visibility: default

Description: `FpSetTimeOfDay` sets the kernel time to the number of seconds since 00:00, January 1 1970, GMT specified in the `tp` record. This time NOT corrected any way, not taking into account time-zones, daylight savings time and so on.

It is simply a wrapper to the kernel system call.

See also: #rtl.unix.FPGetTimeOfDay (1563)

1.4.64 FpSetuid

Synopsis: Set the current user ID

Declaration: `function FpSetuid(uid: TUid) : cint`

Visibility: default

Description: `fpSetUID` sets the user ID of the current process. This call will only work if it is executed as root, or the program is setuid root.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with fpGetErrNo (145).

See also: FpGetGid (146), FpGetUid (148), FpGetEUid (146), FpGetEGid (144), FpGetPid (147), FpGetPPid (148), FpSetGid (167)

1.4.65 FPSigaction

Synopsis: Install signal handler

Declaration: `function FPSigaction(sig: cint;act: psigactionrec;oact: psigactionrec) : cint`

Visibility: default

Description: `FPSigaction` changes the action to take upon receipt of a signal. `Act` and `Oact` are pointers to a SigActionRec (125) record. `Sig` specifies the signal, and can be any signal except **SIGKILL** or **SIGSTOP**.

If `Act` is non-nil, then the new action for signal `Sig` is taken from it. If `OAct` 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 `sig` 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: Extended error information can be retrieved using fpGetErrno (145).

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: FpSigProcMask (172), FpSigPending (172), FpSigSuspend (173), FpKill (149)

Listing: ./bunixex/ex57.pp

```
Program example57;

{ Program to demonstrate the SigAction function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

uses BaseUnix;

Var
  oa,na : PSigActionRec;

Procedure DoSig(sig : cint);cdecl;

begin
  writeln('Receiving signal: ',sig);
end;

begin
  new(na);
  new(oa);
  na^.sa_Handler:=SigActionHandler(@DoSig);
  fillchar(na^.Sa_Mask,sizeof(na^.sa_mask),#0);
  na^.Sa_Flags:=0;
  {$ifdef Linux}                // Linux specific
    na^.Sa_Restorer:=Nil;
```

```
    {$endif}
    if fpSigAction(SigUsr1,na,oa)<>0 then
      begin
      writeln('Error: ',fpgeterrno,'.');
      halt(1);
      end;
    WriteIn ('Send USR1 signal or press <ENTER> to exit');
    readln;
end.
```

1.4.66 FpSigAddSet

Synopsis: Set a signal in a signal set.

Declaration: `function FpSigAddSet(var nset: tsigset;signo: cint) : cint`

Visibility: default

Description: `FpSigAddSet` adds signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: FpSigEmptySet (170), FpSigFillSet (171), FpSigDelSet (170), FpSigIsMember (171)

1.4.67 FpSigDelSet

Synopsis: Remove a signal from a signal set.

Declaration: `function FpSigDelSet(var nset: tsigset;signo: cint) : cint`

Visibility: default

Description: `FpSigDelSet` removes signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: FpSigEmptySet (170), FpSigFillSet (171), FpSigAddSet (170), FpSigIsMember (171)

1.4.68 FpsigEmptySet

Synopsis: Clear all signals from signal set.

Declaration: `function FpsigEmptySet(var nset: tsigset) : cint`

Visibility: default

Description: `FpSigEmptySet` clears all signals from the signal set `nset`.

Errors: None. This function always returns zero.

See also: FpSigFillSet (171), FpSigAddSet (170), FpSigDelSet (170), FpSigIsMember (171)

1.4.69 FpSigFillSet

Synopsis: Set all signals in signal set.

Declaration: `function FpSigFillSet(var nset: tsigset) : cint`

Visibility: default

Description: `FpSigFillSet` sets all signals in the signal set `nset`.

Errors: None. This function always returns zero.

See also: FpSigEmptySet (170), FpSigAddSet (170), FpSigDelSet (170), FpSigIsMember (171)

1.4.70 FpSigIsMember

Synopsis: Check whether a signal appears in a signal set.

Declaration: `function FpSigIsMember(const nset: tsigset;signo: cint) : cint`

Visibility: default

Description: `FpSigIsMember` checks whether `SigNo` appears in the set `nset`. If it is a member, then 1 is returned. If not, zero is returned.

Errors: If an invalid signal number is given, -1 is returned.

See also: FpSigEmptySet (170), FpSigFillSet (171), FpSigAddSet (170), FpSigDelSet (170)

1.4.71 FpSignal

Synopsis: Install signal handler (deprecated)

Declaration:
```
function FpSignal(signum: LongInt;Handler: signalhandler)
                        : signalhandler
```

Visibility: default

Description: `FPSignal` installs a new signal handler (specified by `Handler`) for signal `SigNum`.

This call has a subset of the functionality provided by the FpSigAction (168) call. The return value for `FpSignal` is the old signal handler, or nil on error.

Errors: Extended error information can be retrieved using fpGetErrno (145).

SIG_ERR An error occurred.

See also: FpSigAction (168), FpKill (149)

Listing: ./bunixex/ex58.pp

```
Program example58;

{ Program to demonstrate the Signal function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}
```

```
uses BaseUnix;

Procedure DoSig(sig : cint);cdecl;

begin
   writeln('Receiving signal: ',sig);
end;

begin
   if fpSignal(SigUsr1,SignalHandler(@DoSig))=signalhandler(SIG_ERR) then
     begin
     writeln('Error: ',fpGetErrno,'.');
     halt(1);
     end;
   Writeln ('Send USR1 signal or press <ENTER> to exit');
   readln;
end.
```

1.4.72 FpSigPending

Synopsis: Return set of currently pending signals

Declaration: `function FpSigPending(var nset: tsigset) : cint`

Visibility: default

Description: `fpSigpending` allows the examination of pending signals (which have been raised while blocked.) The signal mask of pending signals is returned.

Errors: None

See also: fpSigAction (168), fpSigProcMask (172), fpSigSuspend (173), fpSignal (171), fpKill (149)

1.4.73 FpSigProcMask

Synopsis: Set list of blocked signals

Declaration:
```
function FpSigProcMask(how: cint;nset: psigset;oset: psigset) : cint
function FpSigProcMask(how: cint;const nset: tsigset;var oset: tsigset)
                       : cint
```

Visibility: default

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 `nset` argument.

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

SIG_SETMASK The list of blocked signals is set so `nset`.

If `oset` is non-nil, then the old set is stored in it.

Errors: `Errno` is used to report errors.

sys_efault `oset` or `nset` point to an adress outside the range of the process.

sys_eintr System call was interrupted.

See also: fpSigAction (168), fpSigPending (172), fpSigSuspend (173), fpKill (149)

1.4.74 FpSigSuspend

Synopsis: Set signal mask and suspend process till signal is received

Declaration:
```
function FpSigSuspend(const sigmask: tsigset) : cint
```

Visibility: default

Description: `fpSigSuspend` temporarily replaces the signal mask for the process with the one given in `SigMask`, and then suspends the process until a signal is received.

Errors: None

See also: fpSigAction (168), fpSigProcMask (172), fpSigPending (172), fpSignal (171), fpKill (149)

1.4.75 FpSleep

Synopsis: Suspend process for several seconds

Declaration:
```
function FpSleep(seconds: cuint) : cuint
```

Visibility: default

Description: `FpSleep` suspends the process till a time period as specified in `seconds` has passed, then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and the return value is the remaining time till the end of the intended period.

If the function returns without error, the return value is zero.

See also: fpPause (159), fpAlarm (133), fpNanoSleep (156)

Listing: ./bunixex/ex73.pp

```
program example73;

{ Program to demonstrate the FpSleep function. }

uses BaseUnix;

Var
  Res : Longint;

begin
  Write('Sleep returned : ');
  Flush(Output);
  Res:=(fpSleep(10));
  Writeln(res);
  If (res<>0) then
    Writeln('Remaining seconds      : ',res);
end.
```

1.4.76 FpStat

Synopsis: Retrieve file information about a file descriptor.

Declaration:
```
function FpStat(path: pChar;var buf: Stat) : cint
function FpStat(path: AnsiString;var buf: Stat) : cint
function FpStat(path: String;var buf: Stat) : cint
```

Visibility: default

Description: `FpFStat` gets information about the file specified in `Path`, and stores it in `Info`, which is of type stat (126). The function returns zero if the call was succesfull, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno (145).

sys_enoent`Path` does not exist.

See also: FpStat (173), FpLStat (152)

Listing: ./bunixex/ex28.pp

```
program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

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 fpstat ('test.fil',info)<>0 then
     begin
       writeln('Fstat failed. Errno : ',fpgeterrno);
       halt (1);
     end;
  writeln;
  writeln ('Result of fstat on file ''test.fil''.');
  writeln ('Inode   : ',info.st_ino);
  writeln ('Mode    : ',info.st_mode);
  writeln ('nlink   : ',info.st_nlink);
  writeln ('uid     : ',info.st_uid);
  writeln ('gid     : ',info.st_gid);
  writeln ('rdev    : ',info.st_rdev);
  writeln ('Size    : ',info.st_size);
  writeln ('Blksize : ',info.st_blksize);
  writeln ('Blocks  : ',info.st_blocks);
  writeln ('atime   : ',info.st_atime);
  writeln ('mtime   : ',info.st_mtime);
  writeln ('ctime   : ',info.st_ctime);
  { Remove file }
  erase (f);
end.
```

1.4.77 fpSymlink

Synopsis: Create a symbolic link

Declaration: `function fpSymlink(oldname: pchar;newname: pchar) : cint`

Visibility: default

Description: `SymLink` makes `NewName` point to the file in `OldName`, 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 zero if the call was succesful, a nonzero value if the call failed.

Errors: Extended error information is returned by the FpGetErrno (145) function.

sys_eperm The filesystem containing oldpath and newpath does not support linking files.

sys_eaccess Write access for the directory containing `Newpath` is disallowed, or one of the directories in `OldPath` or `NewPath` has no search (=execute) permission.

sys_enoent A directory entry in `OldPath` or `NewPath` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in `OldPath` or `NewPath` is 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 another entry.

See also: FpLink (150), FpUnLink (179), FpReadLink (163)

Listing: ./unixex/ex22.pp

```
Program Example22;

{ Program to demonstrate the SymLink and UnLink functions. }

Uses baseunix,Unix;

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 fpSymLink ('test.txt','new.txt')<>0 then
    writeln ('Error when symlinking !');
  { Removing test.txt still leaves new.txt
    Pointing now to a non-existent file ! }
  If fpUnlink ('test.txt')<>0 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 fpUnlink ('new.txt')<>0 then
  WriteLn ('Error when unlinking !');
end.
```

1.4.78 fpS_ISBLK

Synopsis: Is file a block device

Declaration: `function fpS_ISBLK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISBLK` checks the file mode `m` to see whether the file is a block device file. If so it returns `True`.

See also: FpFStat (143), FpS_ISLNK (177), FpS_ISREG (177), FpS_ISDIR (176), FpS_ISCHR (176), FpS_ISFIFO (176), FpS_ISSOCK (178)

1.4.79 fpS_ISCHR

Synopsis: Is file a character device

Declaration: `function fpS_ISCHR(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISCHR` checks the file mode `m` to see whether the file is a character device file. If so it returns `True`.

See also: FpFStat (143), FpS_ISLNK (177), FpS_ISREG (177), FpS_ISDIR (176), FpS_ISBLK (176), FpS_ISFIFO (176), FpS_ISSOCK (178)

1.4.80 fpS_ISDIR

Synopsis: Is file a directory

Declaration: `function fpS_ISDIR(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISDIR` checks the file mode `m` to see whether the file is a directory. If so, it returns `True`

See also: FpFStat (143), FpS_ISLNK (177), FpS_ISREG (177), FpS_ISCHR (176), FpS_ISBLK (176), fpS_ISFIFO (176), FpS_ISSOCK (178)

1.4.81 fpS_ISFIFO

Synopsis: Is file a FIFO

Declaration: `function fpS_ISFIFO(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISFIFO` checks the file mode `m` to see whether the file is a fifo (a named pipe). If so it returns `True`.

See also: FpFStat (143), FpS_ISLNK (177), FpS_ISREG (177), FpS_ISCHR (176), FpS_ISBLK (176), FpS_ISDIR (176), FpS_ISSOCK (178)

1.4.82 fpS_ISLNK

Synopsis: Is file a symbolic link

Declaration: `function fpS_ISLNK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISLNK` checks the file mode `m` to see whether the file is a symbolic link. If so it returns `True`

See also: FpFStat (143), FpS_ISFIFO (176), FpS_ISREG (177), FpS_ISCHR (176), FpS_ISBLK (176), FpS_ISDIR (176), FpS_ISSOCK (178)

Listing: ./bunixex/ex53.pp

```
Program Example53;

{ Program to demonstrate the S_ISLNK function. }

Uses BaseUnix,Unix;

Var Info : Stat;

begin
  if fpLStat (paramstr(1),@info)=0 then
    begin
     if fpS_ISLNK(info.st_mode) then
       Writeln ('File is a link');
     if fpS_ISREG(info.st_mode) then
       Writeln ('File is a regular file');
     if fpS_ISDIR(info.st_mode) then
       Writeln ('File is a directory');
     if fpS_ISCHR(info.st_mode) then
       Writeln ('File is a character device file');
     if fpS_ISBLK(info.st_mode) then
       Writeln ('File is a block device file');
     if fpS_ISFIFO(info.st_mode) then
       Writeln ('File is a named pipe (FIFO)');
     if fpS_ISSOCK(info.st_mode) then
       Writeln ('File is a socket');
    end;
end.
```

1.4.83 fpS_ISREG

Synopsis: Is file a regular file

Declaration: `function fpS_ISREG(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISREG` checks the file mode `m` to see whether the file is a regular file. If so it returns `True`

See also: FpFStat (143), FpS_ISFIFO (176), FpS_ISLNK (177), FpS_ISCHR (176), FpS_ISBLK (176), FpS_ISDIR (176), FPS_ISSOCK (178)

1.4.84 fpS_ISSOCK

Synopsis: Is file a unix socket

Declaration: `function fpS_ISSOCK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISSOCK` checks the file mode `m` to see whether the file is a socket. If so it returns `True`.

See also: FpFStat (143), FpS_ISFIFO (176), FpS_ISLNK (177), FpS_ISCHR (176), FpS_ISBLK (176), FpS_ISDIR (176), FpS_ISREG (177)

1.4.85 fptime

Synopsis: Return the current unix time

Declaration:
```
function FpTime(var tloc: TTime) : TTime
function fptime : time_t
```

Visibility: default

Description: `FpTime` 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. The result is also stored in `tloc`, if it is specified.

Errors: On error, -1 is returned. Extended error information can be retrieved using fpGetErrno (145).

Listing: ./bunixex/ex1.pp

```
Program Example1;

{ Program to demonstrate the fptime function. }

Uses baseunix;

begin
  Write ('Secs past the start of the Epoch (00:00 1/1/1980) : ');
  Writeln (fptime);
end.
```

1.4.86 FpTimes

Synopsis: Return execution times for the current process

Declaration: `function FpTimes(var buffer: tms) : TClock`

Visibility: default

Description: `fpTimes` stores the execution time of the current process and child processes in `buffer`.

The return value (on linux) is the number of clock ticks since boot time. On error, -1 is returned, and extended error information can be retrieved with fpGetErrno (145).

See also: fpUTime (180)

1.4.87 FpUmask

Synopsis: Set file creation mask.

Declaration: `function FpUmask(cmask: TMode) : TMode`

Visibility: default

Description: `fpUmask` changes the file creation mask for the current user to `cmask`. The current mask is returned.

See also: fpChmod (134)

Listing: ./bunixex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Umask function. }

Uses BaseUnix;

begin
  WriteIn ('Old Umask was : ',fpUmask(&111));
  WRiteIn ('New Umask is  : ',&111);
end.
```

1.4.88 FpUname

Synopsis: Return system name.

Declaration: `function FpUname(var name: UtsName) : cint`

Visibility: default

Description: `Uname` gets the name and configuration of the current linux kernel, and returns it in the `name` record.

On success, 0 is returned, on error, -1 is returned.

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpUTime (180)

1.4.89 FpUnlink

Synopsis: Unlink (i.e. remove) a file.

Declaration:
```
function FpUnlink(path: pChar) : cint
function FpUnlink(path: AnsiString) : cint
```

Visibility: default

Description: `FpUnlink` decreases the link count on file `Path`. `Path` can be of type `AnsiString` or `PChar`. If the link count is zero, the file is removed from the disk.

The function returns zero if the call was succesfull, a nonzero value indicates failure.

Note: There exist a portable alterative to erase files: system.erase. Please use fpUnlink only if you are writing Unix specific code. System.erase will work on all operating systems.

For an example, see FpLink (150).

Errors: Extended error information can be retrieved using fpGetErrno (145).

sys_eaccess You have no write access right in the directory containing `Path`, or you have no search permission in one of the directory components of `Path`.

sys_eperm The directory containing pathname has the sticky-bit set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

sys_enoent A component of the path doesn't exist.

sys_enotdir A directory component of the path is not a directory.

sys_eisdir `Path` refers to a directory.

sys_enomem Insufficient kernel memory.

sys_erofs `Path` is on a read-only filesystem.

See also: FpLink (150), FpSymLink (174)

1.4.90 FpUtime

Synopsis: Set access and modification times of a file (touch).

Declaration:
```
function FpUtime(path: pChar;times: pUtimBuf) : cint
function FpUtime(path: AnsiString;times: pUtimBuf) : cint
```

Visibility: default

Description: `FpUtime` sets the access and modification times of the file specified in `Path`. the `times` record contains 2 fields, `actime`, and `modtime`, both of type `time_t` (commonly a 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.

The function returns zero on success, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno (145).

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoent A directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

Other errors may occur, but aren't documented.

See also: FpTime (178), FpChown (135), FpAccess (132)

Listing: ./bunixex/ex25.pp

```
Program Example25;

{ Program to demonstrate the UTime function. }

Uses Dos,BaseUnix,Unix,UnixUtil;

Var utim : utimbuf;
    dow,msec,year,month,day,hour,minute,second : Word;

begin
  { Set access and modification time of executable source }
  GetTime (hour,minute,second,msec);
  GetDate (year,month,day,dow);
  utim.actime:=LocalToEpoch(year,month,day,hour,minute,second);
  utim.modtime:=utim.actime;
  if Fputime('ex25.pp',@utim)<>0 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.
```

1.4.91 FpWait

Synopsis: Wait for a child to exit.

Declaration: `function FpWait(var stat_loc: cint) : TPid`

Visibility: default

Description: `fpWait` suspends the current process and waits for any child to exit or stop due to a signal. It reports the exit status of the exited child in `stat_loc`.

The return value of the function is the process ID of the child that exited, or -1 on error.

Errors: Extended error information can be retrieved using fpgetErrno (145).

See also: fpFork (142), fpExecve (139), fpWaitPid (181)

1.4.92 FpWaitPid

Synopsis: Wait for a process to terminate

Declaration:
```
function FpWaitpid(pid: TPid;stat_loc: pcint;options: cint) : TPid
function FpWaitPid(pid: TPid;var Status: cint;Options: cint) : TPid
```

Visibility: default

Description: `fpWaitPid` waits for a child process with process ID `Pid` to exit. The value of `Pid` can be one of the following:

Pid < -1 Causes `fpWaitPid` to wait for any child process whose process group ID equals the absolute value of `pid`.

Pid = -1 Causes `fpWaitPid` to wait for any child process.

Pid = 0 Causes `fpWaitPid` to wait for any child process whose process group ID equals the one of the calling process.

Pid > 0 Causes `fpWaitPid` to wait for the child whose process ID equals the value of `Pid`.

The `Options` parameter can be used to specify further how `fpWaitPid` behaves:

WNOHANGCauses `fpWaitpid` to return immediately if no child hasexited.

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

__WCLONE Causes `fpWaitPid` also to wait for threads created by the #rtl.linux.Clone (666) call.

The exit status of the process that caused `fpWaitPID` is reported in `stat_loc` or `Status`.

Upon return, it returns the process id of the process that exited, 0 if no process exited, or -1 in case of failure.

For an example, see fpFork (142).

Errors: Extended error information can be retrieved using fpgetErrno (145).

See also: fpFork (142), fpExecve (139), fpWait (181)

1.4.93 FpWrite

Synopsis: Write data to file descriptor

Declaration:
```
function FpWrite(fd: cint;buf: pChar;nbytes: TSize) : TsSize
function FpWrite(fd: cint;const buf;nbytes: TSize) : TsSize
```

Visibility: default

Description: `FpWrite` writes at most `nbytes` bytes from `buf` to file descriptor `fd`.

The function returns the number of bytes actually written, or -1 if an error occurred.

Errors: Extended error information can be retrieved using fpGetErrno (145).

See also: FpOpen (157), FpClose (136), FpRead (161), FpFTruncate (144), FpLSeek (151)

1.4.94 FpWriteV

Synopsis: Vector write: Write from multiple buffers to a file descriptor

Declaration:
```
function FpWriteV(fd: cint;const iov: piovec;iovcnt: cint) : TsSize
```

Visibility: default

Description: `FpWriteV` writes data to file descriptor `fd`. The data is taken from `iovcnt` buffers described by the tiovec (128) buffers pointed to by `iov`. It works like fpWrite (182) only from multiple buffers.

Errors: On error, -1 is returned.

See also: FpReadV (164), FpPWrite (161), FpPRead (161)

1.4.95 FreeShellArgV

Synopsis: Free the result of a CreateShellArgV (131) function

Declaration: `procedure FreeShellArgV(p: ppchar)`

Visibility: default

Description: `FreeShellArgV` frees the memory pointed to by `P`, which was allocated by a call to CreateShellArgV (131).

Errors: None.

See also: CreateShellArgV (131)

1.4.96 wexitStatus

Synopsis: Extract the exit status from the fpWaitPID (181) result.

Declaration: `function wexitStatus(Status: cint) : cint`

Visibility: default

Description: `WEXITSTATUS` can be used to extract the exit status from `Status`, the result of the FpWaitPID (181) call.

See also: FpWaitPID (181), WTERMSIG (184), WSTOPSIG (183), WIFEXITED (183), WIFSIGNALED (183)

1.4.97 wifexited

Synopsis: Check whether the process exited normally

Declaration: `function wifexited(Status: cint) : Boolean`

Visibility: default

Description: `WIFEXITED` checks `Status` and returns `True` if the status indicates that the process terminated normally, i.e. was not stopped by a signal.

See also: FpWaitPID (181), WTERMSIG (184), WSTOPSIG (183), WIFSIGNALED (183), WEXITSTATUS (183)

1.4.98 wifsignaled

Synopsis: Check whether the process was exited by a signal.

Declaration: `function wifsignaled(Status: cint) : Boolean`

Visibility: default

Description: `WIFSIGNALED` returns `True` if `Status` indicates that the process exited because it received a signal.

See also: FpWaitPID (181), WTERMSIG (184), WSTOPSIG (183), WIFEXITED (183), WEXITSTATUS (183)

1.4.99 wstopsig

Synopsis: Return the exit code from the process.

Declaration: `function wstopsig(Status: cint) : cint`

Visibility: default

Description: `WSTOPSIG` is an alias for WEXITSTATUS (183).

See also: FpWaitPID (181), WTERMSIG (184), WIFEXITED (183), WIFSIGNALED (183), WEXITSTATUS (183)

1.4.100 wtermsig

Synopsis: Return the signal that caused a process to exit.

Declaration: `function wtermsig(Status: cint) : cint`

Visibility: default

Description: `WTERMSIG` extracts from `Status` the signal number which caused the process to exit.

See also: FpWaitPID (181), WSTOPSIG (183), WIFEXITED (183), WIFSIGNALED (183), WEXITSTATUS (183)

Chapter 2

Reference for unit 'Classes'

2.1 Used units

Table 2.1: Used units by unit 'Classes'

Name	Page
rtlconsts	185
sysutils	1350
types	185
typinfo	1504

2.2 Overview

This documentation describes the FPC classes unit. The Classes unit contains basic classes for the Free Component Library (FCL):

- a TList (300) class for maintaining lists of pointers,
- TStringList (345) for lists of strings,
- TCollection (260) to manage collections of objects
- TStream (332) classes to support streaming.

Furthermore it introduces methods for object persistence, and classes that understand an owner-owned relationship, with automatic memory management.

2.3 Constants, types and variables

2.3.1 Constants

```
BITSHIFT = 5
```

Used to calculate the size of a bits array

```
dupAccept = Types.dupAccept
```

Duplicate values can be added to the list.

```
dupError = Types.dupError
```

If an attempt is made to add a duplicate value to the list, an EStringListError (217) exception is raised.

```
dupIgnore = Types.dupIgnore
```

Duplicate values will not be be added to the list, but no error will be triggered.

```
FilerSignature : Array[1..4] of Char = 'TPF0'
```

Constant that is found at the start of a binary stream containing a streamed component.

```
fmCreate = $FFFF
```

TFileStream.Create (284) creates a new file if needed.

```
fmOpenRead = 0
```

TFileStream.Create (284) opens a file with read-only access.

```
fmOpenReadWrite = 2
```

TFileStream.Create (284) opens a file with read-write access.

```
fmOpenWrite = 1
```

TFileStream.Create (284) opens a file with write-only access.

```
MASK = 31
```

Bitmask with all bits on.

```
MaxBitFlags = MaxBitRec * 32
```

Maximum number of bits in TBits collection.

```
MaxBitRec = $FFFF div ( SizeOf ( longint ) )
```

Maximum number of bit records in TBits.

```
MaxListSize = Maxint div 16
```

This constant sets the maximum number of elements in a TList (300).

```
scAlt = $8000
```

Indicates ALT key in a keyboard shortcut.

```
scCtrl = $4000
```

indicates CTRL key in a keyboard shortcut.

```
scNone = 0
```

Indicates no special key is presed in a keyboard shortcut.

```
scShift = $2000
```

Indicates Shift key in a keyboard shortcut.

```
soFromBeginning = 0
```

Seek (333) starts relative to the stream origin.

```
soFromCurrent = 1
```

Seek (333) starts relative to the current position in the stream.

```
soFromEnd = 2
```

Seek (333) starts relative to the stream end.

```
toEOF = Char ( 0 )
```

Value returned by TParser.Token (316) when the end of the input stream was reached.

```
toFloat = Char ( 4 )
```

Value returned by TParser.Token (316) when a floating point value was found in the input stream.

```
toInteger = Char ( 3 )
```

Value returned by TParser.Token (316) when an integer was found in the input stream.

```
toString = Char ( 2 )
```

Value returned by TParser.Token (316) when a string was found in the input stream.

```
toSymbol = Char ( 1 )
```

Value returned by TParser.Token (316) when a symbol was found in the input stream.

```
toWString = Char ( 5 )
```

Value returned by TParser.Token (316) when a widestring was found in the input stream.

2.3.2 Types

```
HMODULE = LongInt
```

FPC doesn't support modules yet, so this is a dummy type.

```
HRSRC = LongInt
```

This type is provided for Delphi compatibilty, it is used for resource streams.

```
PPointerList = ^TPointerList
```

Pointer to an array of pointers.

```
PStringItem = ^TStringItem
```

Pointer to a TStringItem (197) record.

```
PStringItemList = ^TStringItemList
```

Pointer to a TStringItemList (197).

```
TActiveXRegType = (axrComponentOnly,axrIncludeDescendants)
```

Table 2.2: Enumeration values for type TActiveXRegType

Value	Explanation
axrComponentOnly	
axrIncludeDescendants	

This type is provided for compatibility only, and is currently not used in Free Pascal.

```
TAlignment = (taLeftJustify,taRightJustify,taCenter)
```

Table 2.3: Enumeration values for type TAlignment

Value	Explanation
taCenter	Text is displayed centered.
taLeftJustify	Text is displayed aligned to the left
taRightJustify	Text is displayed aligned to the right.

The `TAlignment` type is used to specify the alignment of the text in controls that display a text.

```
TAncestorNotFoundEvent = procedure(Reader: TReader;
                                   const ComponentName: String;
                                   ComponentClass: TPersistentClass;
                                   var Component: TComponent) of object
```

This event occurs when an ancestor component cannot be found.

```
TBasicActionClass = Class of TBasicAction
```

TBasicAction (237) class reference.

```
TBasicActionLinkClass = Class of TBasicActionLink
```

TBasicActionLink (241) class reference.

```
TBiDiMode = (bdLeftToRight,bdRightToLeft,bdRightToLeftNoAlign,
             bdRightToLeftReadingOnly)
```

Table 2.4: Enumeration values for type TBiDiMode

Value	Explanation
bdLeftToRight	Texts read from left to right.
bdRightToLeft	Texts read from right to left.
bdRightToLeftNoAlign	Texts read from right to left, but not right-aligned
bdRightToLeftReadingOnly	Texts read from right to left

`TBiDiMode` describes bi-directional support for displaying texts.

```
TBitArray = Array[0..MaxBitRec-1] of cardinal
```

Array to store bits.

```
TCollectionItemClass = Class of TCollectionItem
```

`TCollectionItemClass` is used by the TCollection.ItemClass (264) property of TCollection (260) to identify the descendent class of TCollectionItem (265) which should be created and managed.

```
TCollectionNotification = (cnAdded,cnExtracting,cnDeleting)
```

Table 2.5: Enumeration values for type TCollectionNotification

Value	Explanation
cnAdded	An item is added to the collection.
cnDeleting	An item is deleted from the collection.
cnExtracting	An item is extracted from the collection.

`TCollectionNotification` is used in the TCollection (260) class to send notifications about changes to the collection.

```
TComponentClass = Class of TComponent
```

The `TComponentClass` type is used when constructing TComponent (267) descendent instances and when registering components.

```
TComponentName = String
```

Names of components are of type `TComponentName`. By specifying a different type, the Object inspector can handle this property differently than a standard string property.

```
TComponentState= Set of (csLoading,csReading,csWriting,csDestroying,
                         csDesigning,csAncestor,csUpdating,csFixups,
                         csFreeNotification,csInline,csDesignInstance)
```

Indicates the state of the component during the streaming process.

```
TComponentStyle= Set of (csInheritable,csCheckPropAvail,csSubComponent,
                         csTransient)
```

Describes the style of the component.

```
TCreateComponentEvent = procedure(Reader: TReader;
                                  ComponentClass: TComponentClass;
                                  var Component: TComponent) of object
```

Event handler type, occurs when a component instance must be created when a component is read from a stream.

```
TDuplicates = Types.TDuplicates
```

Type to describe what to do with duplicate values in a TStringlist (345).

```
TFilerFlag = (ffInherited,ffChildPos,ffInline)
```

Table 2.6: Enumeration values for type TFilerFlag

Value	Explanation
ffChildPos	The position of the child on it's parent is included.
ffInherited	Stored object is an inherited object.
ffInline	Used for frames.

The `TFiler` class uses this enumeration type to decide whether the streamed object was streamed as part of an inherited form or not.

```
TFilerFlags= Set of (ffChildPos,ffInherited,ffInline)
```

Set of TFilerFlag (190)

```
TFindAncestorEvent = procedure(Writer: TWriter;Component: TComponent;
                               const Name: String;
                               var Ancestor: TComponent;
                               var RootAncestor: TComponent) of object
```

Event that occurs w

```
TFindComponentClassEvent = procedure(Reader: TReader;
                                     const ClassName: String;
                                     var ComponentClass: TComponentClass)
                                      of object
```

Event handler type, occurs when a component class pointer must be found when reading a component from a stream.

```
TFindGlobalComponent = function(const Name: String) : TComponent
```

`TFindGlobalComponent` is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

The variable FindGlobalComponent (202) is a callback of type `TFindGlobalComponent`. It can be set by the IDE when an unknown reference is found, to offer the designer to redirect the link to a new component.

```
TFindMethodEvent = procedure(Reader: TReader;const MethodName: String;
                             var Address: Pointer;var Error: Boolean)
                              of object
```

If a TReader (318) instance needs to locate a method and it doesn't find it in the streamed form, then the OnFindMethod (327) event handler will be called, if one is installed. This event can be assigned in order to use different locating methods. If a method is found, then its address should be returned in `Address`. The `Error` should be set to `True` if the reader should raise an exception after the event was handled. If it is set to `False` no exception will be raised, even if no method was found. On entry, `Error` will be set to `True`.

```
TGetChildProc = procedure(Child: TComponent) of object
```

Callback used when obtaining child components.

```
TGetStrProc = procedure(const S: String) of object
```

This event is used as a callback to retrieve string values. It is used, among other things, to pass along string properties in property editors.

```
THandle = System.THandle
```

This type is used as the handle for THandleStream (292) stream descendents

```
THelpContext = -MaxLongint..MaxLongint
```

Range type to specify help contexts.

```
THelpEvent = function(Command: Word;Data: LongInt;var CallHelp: Boolean)
                      : Boolean of object
```

This event is used for display of online help.

```
THelpType = (htKeyword,htContext)
```

Table 2.7: Enumeration values for type THelpType

Value	Explanation
htContext	Help type: Context ID help.
htKeyword	Help type: Keyword help

Enumeration type specifying the kind of help requested.

```
TIdentMapEntry = record
  Value : Integer;
  Name : String;
end
```

`TIdentMapEntry` is used internally by the IdentToInt (205) and IntToIdent (206) calls to store the mapping between the identifiers and the integers they represent.

```
TIdentToInt = function(const Ident: String;var Int: LongInt) : Boolean
```

`TIdentToInt` is a callback used to look up identifiers (`Ident`) and return an integer value corresponding to this identifier (`Int`). The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIdentToInt` should be specified when an integer is registered using the RegisterIntegerConsts (211) call.

```
TInitComponentHandler = function(Instance: TComponent;
                                 RootAncestor: TClass) : Boolean
```

`TInitComponentHandler` is a callback type. It is used in the InitInheritedComponent (**??**) call to initialize a component. Callbacks of this type are registered with the RegisterInitComponentHandler (211) call.

```
TIntToIdent = function(Int: LongInt;var Ident: String) : Boolean
```

`TIdentToInt` is a callback used to look up integers (`Ident`) and return an identifier (`Ident`) that can be used to represent this integer value in an IDE. The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIntToIdent` should be specified when an integer is registered using the RegisterIntegerConsts (211) call.

```
TLeftRight = ..taRightJustify
```

`TLeftRight` is a subrange type based on the TAlignment (188) enumerated type. It contains only the left and right alignment constants.

```
TListAssignOp = (laCopy,laAnd,laOr,laXor,laSrcUnique,laDestUnique)
```

This type determines what operation TList.Assign (304) or TFPList.assign (290) performs.

```
TListCallback = Types.TListCallback
```

Table 2.8: Enumeration values for type TListAssignOp

Value	Explanation
laAnd	Remove all elements not first second list
laCopy	Clear list and copy all strings from second list.
laDestUnique	Keep all elements that exists only in list2
laOr	Add all elements from second (and optional third) list, eliminate duplicates
laSrcUnique	Just keep all elements that exist only in source list
laXor	Remove elements in second lists, Add all elements from second list not in first list

`TListCallback` is the method callback prototype for the function that is passed to the TFPList.ForEachCall (291) call. The `data` argument will be filled with all the pointers in the list (one per call) and the `arg` argument is the `Arg` argument passed to the `ForEachCall` call.

```
TListNotification = (lnAdded,lnExtracted,lnDeleted)
```

Table 2.9: Enumeration values for type TListNotification

Value	Explanation
lnAdded	List change notification: Element added to the list.
lnDeleted	List change notification: Element deleted from the list.
lnExtracted	List change notification: Element extracted from the list.

Kind of list notification event.

```
TListSortCompare = function(Item1: Pointer;Item2: Pointer) : Integer
```

Callback type for the list sort algorithm.

```
TListStaticCallback = Types.TListStaticCallback
```

`TListCallback` is the procedurald callback prototype for the function that is passed to the TFPList.ForEachCall (291) call. The `data` argument will be filled with all the pointers in the list (one per call) and the `arg` argument is the `Arg` argument passed to the `ForEachCall` call.

```
TNotifyEvent = procedure(Sender: TObject) of object
```

Most event handlers are implemented as a property of type `TNotifyEvent`. When this is set to a certain method of a class, when the event occurs, the method will be called, and the class that generated the event will pass itself along as the `Sender` argument.

```
TOperation = (opInsert,opRemove)
```

Operation of which a component is notified.

```
TPersistentClass = Class of TPersistent
```

`TPersistentClass` is the class reference type for the TPersistent (317) class.

Table 2.10: Enumeration values for type TOperation

Value	Explanation
opInsert	A new component is being inserted in the child component list.
opRemove	A component is being removed from the child component list.

```
TPoint = Types.TPoint
```

This record describes a coordinate. It is used to handle the Top (267) and Left (267) properties of TComponent (267).

`X` represents the X-Coordinate of the point described by the record. `Y` represents the Y-Coordinate of the point described by the record.

```
TPointerList = Array[0..MaxListSize-1] of Pointer
```

Type for an Array of pointers.

```
TPropertyNotFoundEvent = procedure(Reader: TReader;
                                   Instance: TPersistent;
                                   var PropName: String;IsPath: Boolean;
                                   var Handled: Boolean;
                                   var Skip: Boolean) of object
```

`TPropertyNotFoundEvent` is the prototype for the TReader.OnPropertyNotFound (327) event. `Reader` is the sender of the event, `Instance` is the instance that is being streamed. `PropInfo` is a pointer to the RTTI information for the property being read. `Handled` should be set to `True` if the handler redirected the unknown property succesfully, and `Skip` should be set to `True` if the value should be skipped. `IsPath` determines whether the property refers to a sub-property.

```
TReadComponentsProc = procedure(Component: TComponent) of object
```

Callback type when reading a component from a stream

```
TReaderError = procedure(Reader: TReader;const Message: String;
                         var Handled: Boolean) of object
```

Event handler type, called when an error occurs during the streaming.

```
TReaderProc = procedure(Reader: TReader) of object
```

The `TReaderProc` reader procedure is a callback procedure which will be used by a TPersistent (317) descendent to read user properties from a stream during the streaming process. The `Reader` argument is the writer object which can be used read properties from the stream.

```
TReadWriteStringPropertyEvent = procedure(Sender: TObject;
                                          const Instance: TPersistent;
                                          PropInfo: PPropInfo;
                                          var Content: String) of object
```

TReadWriteStringPropertyEvent is the prototype for the TReader.OnReadStringProperty (329) event handler. Reader is the sender of the event, Instance is the instance that is being streamed. PropInfo is a pointer to the RTTI information for the property being read. Content is the string as it was read from the stream.

```
TRect = Types.TRect
```

TRect describes a rectangle in space with its upper-left (in (Top,Left>)) and lower-right (in (Bottom ,Right)) corners.

```
TReferenceNameEvent = procedure(Reader: TReader;var Name: String)
                                  of object
```

Occurs when a named object needs to be looked up.

```
TSeekOrigin = (soBeginning,soCurrent,soEnd)
```

Table 2.11: Enumeration values for type TSeekOrigin

Value	Explanation
soBeginning	Offset is interpreted relative to the start of the stream.
soCurrent	Offset is interpreted relative to the current position in the stream.
soEnd	Offset is interpreted relative to the end of the stream.

Specifies the origin of the TStream.Seek (333) method.

```
TSetMethodPropertyEvent = procedure(Reader: TReader;
                                    Instance: TPersistent;
                                    PropInfo: PPropInfo;
                                    const TheMethodName: String;
                                    var Handled: Boolean) of object
```

TSetMethodPropertyEvent is the prototype for the TReader.OnSetMethodProperty (328) event. Reader is the sender of the event, Instance is the instance that is being streamed. PropInfo is a pointer to the RTTI information for the property being read, and TheMethodName is the name of the method that the property should be set to. Handled should be set to True if the handler set the property succesfully.

```
TSetNameEvent = procedure(Reader: TReader;Component: TComponent;
                          var Name: String) of object
```

Occurs when the reader needs to set a component's name.

```
TShiftState= Set of (ssShift,ssAlt,ssCtrl,ssLeft,ssRight,ssMiddle,
                     ssDouble,ssMeta,ssSuper,ssHyper,ssAltGr,ssCaps,
                     ssNum,ssScroll,ssTriple,ssQuad,ssExtra1,ssExtra2)
```

This type is used when describing a shortcut key or when describing what special keys are pressed on a keyboard when a key event is generated.

The set contains the special keys that can be used in combination with a 'normal' key.

Table 2.12: Enumeration values for type TShiftStateEnum

Value	Explanation
ssAlt	Alt key pressed
ssAltGr	Alt-GR key pressed.
ssCaps	Caps lock key pressed
ssCtrl	Ctrl key pressed
ssDouble	Double mouse click.
ssExtra1	Extra key 1
ssExtra2	Extra key 2
ssHyper	Hyper key pressed.
ssLeft	Left mouse button pressed.
ssMeta	Meta key pressed.
ssMiddle	Middle mouse button pressed.
ssNum	Num lock key pressed
ssQuad	Quadruple mouse click
ssRight	Right mouse button pressed.
ssScroll	Scroll lock key pressed
ssShift	Shift key pressed
ssSuper	Super key pressed.
ssTriple	Triple mouse click

```
TShiftStateEnum = (ssShift,ssAlt,ssCtrl,ssLeft,ssRight,ssMiddle,
                   ssDouble,ssMeta,ssSuper,ssHyper,ssAltGr,ssCaps,ssNum,
                   ssScroll,ssTriple,ssQuad,ssExtra1,ssExtra2)
```

Keyboard/Mouse shift state enumerator

```
TShortCut =  ( Word )..High ( Word )
```

Enumeration type to identify shortcut key combinations.

```
TSmallPoint = record
  x : SmallInt;
  y : SmallInt;
end
```

Same as TPoint (194), only the X and Y ranges are limited to 2-byte integers instead of 4-byte integers.

```
TStreamOwnership = (soReference,soOwned)
```

Table 2.13: Enumeration values for type TStreamOwnership

Value	Explanation
soOwned	Stream is owned: it will be freed when the adapter is freed.
soReference	Stream is referenced only, it is not freed by the adapter

The ownership of a streamadapter determines what happens with the stream on which a TStreamAdapter (340) acts, when the adapter is freed.

```
TStreamProc = procedure(Stream: TStream) of object
```

Procedure type used in streaming.

```
TStringItem = record
  FString : String;
  FObject : TObject;
end
```

The `TStringItem` is used to store the string and object items in a TStringList (345) string list instance. It should never be used directly.

```
TStringItemList = Array[0..MaxListSize] of TStringItem
```

This declaration is provided for Delphi compatibility, it is not used in Free Pascal.

```
TStringListSortCompare = function(List: TStringList;Index1: Integer;
                                  Index2: Integer) : Integer
```

Callback type used in stringlist compares.

```
TSynchronizeProcVar = procedure
```

Synchronize callback type

```
TThreadMethod = procedure of object
```

Procedure variable used when synchronizing threads.

```
TThreadPriority = (tpIdle,tpLowest,tpLower,tpNormal,tpHigher,tpHighest,
                   tpTimeCritical)
```

Table 2.14: Enumeration values for type TThreadPriority

Value	Explanation
tpHigher	Thread runs at high priority
tpHighest	Thread runs at highest possible priority.
tpIdle	Thread only runs when other processes are idle.
tpLower	Thread runs at a lower priority.
tpLowest	Thread runs at the lowest priority.
tpNormal	Thread runs at normal process priority.
tpTimeCritical	Thread runs at realtime priority.

Enumeration specifying the priority at which a thread runs.

```
TValueType = (vaNull,vaList,vaInt8,vaInt16,vaInt32,vaExtended,vaString,
              vaIdent,vaFalse,vaTrue,vaBinary,vaSet,vaLString,vaNil,
              vaCollection,vaSingle,vaCurrency,vaDate,vaWString,vaInt64,
              vaUTF8String)
```

Table 2.15: Enumeration values for type TValueType

Value	Explanation
vaBinary	Binary data follows.
vaCollection	Collection follows
vaCurrency	Currency value follows
vaDate	Date value follows
vaExtended	Extended value.
vaFalse	Boolean `False` value.
vaIdent	Identifier.
vaInt16	Integer value, 16 bits long.
vaInt32	Integer value, 32 bits long.
vaInt64	Integer value, 64 bits long.
vaInt8	Integer value, 8 bits long.
vaList	Identifies the start of a list of values
vaLString	Ansistring data follows.
vaNil	Nil pointer.
vaNull	Empty value. Ends a list.
vaSet	Set data follows.
vaSingle	Single type follows.
vaString	String value.
vaTrue	Boolean `True` value.
vaUTF8String	UTF8 encoded unicode string.
vaWString	Widestring value follows.

Enumerated type used to identify the kind of streamed property

```
TWriteMethodPropertyEvent = procedure(Writer: TWriter;
                                      Instance: TPersistent;
                                      PropInfo: PPropInfo;
                                      const MethodValue: TMethod;
                                      const DefMethodValue: TMethod;
                                      var Handled: Boolean) of object
```

`TWriteMethodPropertyEvent` is the prototype for the TWriter.OnWriteMethodProperty (377) event. `Writer` is the sender of the event, `Instance` is the instance that is being streamed. `PropInfo` is a pointer to the RTTI information for the property being written, and `MethodValue` is the value of the method that the property was set to. `DefMethodCodeValue` is set to the default value of the property (Nil or the parent value). `Handled` should be set to `True` if the handler set the property succesfully.

```
TWriterProc = procedure(Writer: TWriter) of object
```

The `TWriterProc` writer procedure is a callback procedure which will be used by a TPersistent (317) descendent to write user properties from a stream during the streaming process. The `Writer` argument is the writer object which can be used write properties to the stream.

2.3.3 Variables

```
AddDataModule : procedure(DataModule: TDataModule) of object
```

`AddDataModule` can be set by an IDE or a streaming mechanism to receive notification when a new instance of a TDataModule (279) descendent is created.

```
ApplicationHandleException : procedure(Sender: TObject) of object
```

`ApplicationHandleException` can be set by an application object to handle any exceptions that may occur when a TDataModule (279) is created.

```
ApplicationShowException : procedure(E: Exception) of object
```

Unused.

```
GlobalNameSpace : IReadWriteSync
```

An interface protecting the global namespace. Used when reading/writing to the global namespace list during streaming of forms.

```
MainThreadID : TThreadID
```

ID of main thread. Unused at this point.

```
RegisterComponentsProc : procedure(const Page: String;ComponentClasses: Array of TCo
```

`RegisterComponentsProc` can be set by an IDE to be notified when new components are being registered. Application programmers should never have to set `RegisterComponentsProc`

```
RegisterNoIconProc : procedure(ComponentClasses: Array of TComponentClass)
```

`RegisterNoIconProc` can be set by an IDE to be notified when new components are being registered, and which do not need an Icon in the component palette. Application programmers should never have to set `RegisterComponentsProc`

```
RemoveDataModule : procedure(DataModule: TDataModule) of object
```

`RemoveDataModule` can be set by an IDE or a streaming mechanism to receive notification when an instance of a TDataModule (279) descendent is freed.

```
WakeMainThread : TNotifyEvent = nil
```

`WakeMainThread` is called by the TThread.synchronize (366) call. It should alert the main program thread that a thread is waiting for synchronization. The call is executed by the thread, and should therefore NOT synchronize the thread, but should somehow signal the main thread that a thread is waiting for synchronization. For example, by sending a message.

2.4 Procedures and functions

2.4.1 ActivateClassGroup

Synopsis: Activates a class group

Declaration: `function ActivateClassGroup(AClass: TPersistentClass) : TPersistentClass`

Visibility: default

Description: `ActivateClassGroup` activates the group of classes to which `AClass` belongs. The function returns the class that was last used to activate the class group.

The class registration and streaming mechanism allows to organize the classes in groups. This allows an IDE to form groups of classes, which can be enabled or disabled. It is not needed at Run-Time.

Errors: If `AClass` does not belong to a class group, an exception is raised.

See also: StartClassGroup (213), GroupDescendentsWith (204), ClassGroupOf (201)

2.4.2 BeginGlobalLoading

Synopsis: Not yet implemented

Declaration: `procedure BeginGlobalLoading`

Visibility: default

Description: Not yet implemented

2.4.3 BinToHex

Synopsis: Convert a binary buffer to a hexadecimal string

Declaration: `procedure BinToHex(BinValue: PChar;HexValue: PChar;BinBufSize: Integer)`

Visibility: default

Description: `BinToHex` converts the byte values in `BinValue` to a string consisting of 2-charachter hexadecimal strings in `HexValue`. `BufSize` specifies the length of `BinValue`, which means that `HexValue` must have size `2*BufSize`.

For example a buffer containing the byte values 255 and 0 will be converted to FF00.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: HexToBin (204)

2.4.4 Bounds

Synopsis: Returns a `TRect` structure with the bounding rect of the given location and size.

Declaration:
```
function Bounds(ALeft: Integer;ATop: Integer;AWidth: Integer;
                AHeight: Integer) : TRect
```

Visibility: default

Description: `Bounds` returns a TRect (195) record with the given origin `(ALeft,ATop)` and dimensions `(AWidth,AHeight)` filled in.

2.4.5 CheckSynchronize

Synopsis: Check whether there are any synchronize calls in the synchronize queue.

Declaration:
```
function CheckSynchronize(timeout: LongInt) : Boolean
```

Visibility: default

Description: `CheckSynchronize` should be called regularly by the main application thread to handle any TThread.synchronize (366) calls that may be waiting for execution by the main thread.

See also: TThread.synchronize (366)

2.4.6 ClassGroupOf

Synopsis: Returns the class group to which an instance or class belongs

Declaration:
```
function ClassGroupOf(AClass: TPersistentClass) : TPersistentClass
function ClassGroupOf(Instance: TPersistent) : TPersistentClass
```

Visibility: default

Description: `ClassGroupOf` returns the class group to which `AClass` or `Instance` belongs.

Errors: The result is `Nil` if no matching class group is found.

See also: StartClassGroup (213), ActivateClassGroup (199), GroupDescendentsWith (204)

2.4.7 CollectionsEqual

Synopsis: Returns `True` if two collections are equal.

Declaration:
```
function CollectionsEqual(C1: TCollection;C2: TCollection) : Boolean
function CollectionsEqual(C1: TCollection;C2: TCollection;
                          Owner1: TComponent;Owner2: TComponent)
                           : Boolean
```

Visibility: default

Description: `CollectionsEqual` is not yet implemented. It simply returns `False`

2.4.8 EndGlobalLoading

Synopsis: Not yet implemented.

Declaration:
```
procedure EndGlobalLoading
```

Visibility: default

Description: Not yet implemented.

2.4.9 ExtractStrings

Synopsis: Split a string in different words.

Declaration:
```
function ExtractStrings(Separators: TSysCharSet;WhiteSpace: TSysCharSet;
                        Content: PChar;Strings: TStrings) : Integer
```

Visibility: default

Description: `ExtractStrings` splits `Content` (a null-terminated string) into words, and adds the words to the `Strings` stringlist. The words are separated by `Separators` and any characters in `whitespace` are stripped from the strings. The space and CR/LF characters are always considered whitespace.

Errors: No length checking is performed on `Content`. If no null-termination character is present, an access violation may occur. Likewise, if `Strings` is not valid, an access violation may occur.

2.4.10 FindClass

Synopsis: Returns the class pointer of a class with given name.

Declaration:
```
function FindClass(const AClassName: String) : TPersistentClass
```

Visibility: default

Description: `FindClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, an exception is raised.

The GetClass (203) function does not raise an exception when it does not find the class, but returns a `Nil` pointer instead.

See also: RegisterClass (209), GetClass (203)

2.4.11 FindGlobalComponent

Synopsis: Callback used when a component must be found.

Declaration:
```
function FindGlobalComponent(const Name: String) : TComponent
```

Visibility: default

Description: `FindGlobalComponent` is a callback of type TFindGlobalComponent (191). It can be set by the IDE when an unknown reference is found, to offer the user to redirect the link to a new component.

It is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

See also: TFindGlobalComponent (191)

2.4.12 FindIdentToInt

Synopsis: Return the string to integer converter for an integer type

Declaration:
```
function FindIdentToInt(AIntegerType: Pointer) : TIdentToInt
```

Visibility: default

Description: `FindIdentToInt` returns the handler that handles the conversion of a string representation to an integer that can be used in component streaming, when IdentToInt (205) is called.

Errors: Nil is returned if no handler is registered for the given type.

2.4.13 FindIntToIdent

Synopsis: Return the integer to string converter for an integer type

Declaration:
```
function FindIntToIdent(AIntegerType: Pointer) : TIntToIdent
```

Visibility: default

Description: `FindIntToIdent` returns the handler that handles the conversion of an integer to a string representation that can be used in component streaming, when IntToIdent (206) is called.

Errors: Nil is returned if no handler is registered for the given type.

See also: IntToIdent (206), TIntToIdent (192), FindIdentToInt (202)

2.4.14 FindNestedComponent

Synopsis: Finds the component with name path starting at the indicated root component.

Declaration:
```
function FindNestedComponent(Root: TComponent;APath: String;
                             CStyle: Boolean) : TComponent
```

Visibility: default

Description: `FindNestedComponent` will descend through the list of owned components (starting at `Root`) and will return the component whose name path matches `NamePath`. As a path separator the characters . (dot), - (dash) and > (greater than) can be used

See also: GlobalFixupReferences (204)

2.4.15 GetClass

Synopsis: Returns the class pointer of a class with given name.

Declaration:
```
function GetClass(const AClassName: String) : TPersistentClass
```

Visibility: default

Description: `GetClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, `Nil` is returned.

The FindClass (202) function will raise an exception if the does not find the class.

See also: RegisterClass (209), GetClass (203)

2.4.16 GetFixupInstanceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component, whose reference contains `ReferenceRootName`

Declaration:
```
procedure GetFixupInstanceNames(Root: TComponent;
                                const ReferenceRootName: String;
                                Names: TStrings)
```

Visibility: default

Description: `GetFixupInstanceNames` examines the list of unresolved references and returns the names of classes that contain unresolved references to the `Root` component in the list `Names`. The list is not cleared prior to filling it.

See also: GetFixupReferenceNames (204), GlobalFixupReferences (204)

2.4.17 GetFixupReferenceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component.

Declaration:
```
procedure GetFixupReferenceNames(Root: TComponent;Names: TStrings)
```

Visibility: default

Description: `GetFixupReferenceNames` examines the list of unresolved references and returns the names of properties that must be resolved for the component `Root` in the list `Names`. The list is not cleared prior to filling it.

See also: GetFixupInstanceNames (203), GlobalFixupReferences (204)

2.4.18 GlobalFixupReferences

Synopsis: Called to resolve unresolved references after forms are loaded.

Declaration:
```
procedure GlobalFixupReferences
```

Visibility: default

Description: `GlobalFixupReferences` runs over the list of unresolved references and tries to resolve them. This routine should under normal circumstances not be called in an application programmer's code. It is called automatically by the streaming system after a component has been instantiated and its properties read from a stream. It will attempt to resolve references to other global components.

See also: GetFixupReferenceNames (204), GetFixupInstanceNames (203)

2.4.19 GroupDescendentsWith

Synopsis: Add class to the group of another class.

Declaration:
```
procedure GroupDescendentsWith(AClass: TPersistentClass;
                               AClassGroup: TPersistentClass)
```

Visibility: default

Description: `GroupDescendentsWith` adds `AClass` to the group that `AClassGroup` belongs to. If `AClassGroup` belongs to more than 1 group, then it is added to the group which contains the nearest ancestor.

The class registration and streaming mechanism allows to organize the classes in groups. This allows an IDE to form groups of classes, which can be enabled or disabled. It is not needed at Run-Time.

Errors:

See also: StartClassGroup (213), ActivateClassGroup (199), ClassGroupOf (201)

2.4.20 HexToBin

Synopsis: Convert a hexadecimal string to a binary buffer

Declaration:
```
function HexToBin(HexValue: PChar;BinValue: PChar;BinBufSize: Integer)
                  : Integer
```

Visibility: default

Description: `HexToBin` scans the hexadecimal string representation in `HexValue` and transforms every 2 character hexadecimal number to a byte and stores it in `BinValue`. The buffer size is the size of the binary buffer. Scanning will stop if the size of the binary buffer is reached or when an invalid character is encountered. The return value is the number of stored bytes.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: BinToHex (200)

2.4.21 IdentToInt

Synopsis: Looks up an integer value in a integer-to-identifier map list.

Declaration:
```
function IdentToInt(const Ident: String;var Int: LongInt;
                    const Map: Array of TIdentMapEntry) : Boolean
```

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Name` field matches `Ident` and returns the corresponding integer value in `Int`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: TIdentToInt (192), TIntToIdent (192), IntToIdent (206), TIdentMapEntry (192)

2.4.22 InitComponentRes

Synopsis: Provided for Delphi compatibility only

Declaration:
```
function InitComponentRes(const ResName: String;Instance: TComponent)
                          : Boolean
```

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `false`.

See also: ReadComponentRes (208)

2.4.23 InitInheritedComponent

Synopsis: Initializes a component descending from `RootAncestor`

Declaration:
```
function InitInheritedComponent(Instance: TComponent;
                                RootAncestor: TClass) : Boolean
```

Visibility: default

Description: `InitInheritedComponent` should be called from a constructor to read properties of the component `Instance` from the streaming system. The `RootAncestor` class is the root class from which `Instance` is a descendent. This must be one of `TDatamodule`, `TCustomForm` or `TFrame`.

The function returns `True` if the properties were successfully read from a stream or `False` if some error occurred.

See also: ReadComponentRes (208), ReadComponentResEx (208), ReadComponentResFile (208)

2.4.24 IntToIdent

Synopsis: Looks up an identifier for an integer value in a identifier-to-integer map list.

Declaration:
```
function IntToIdent(Int: LongInt;var Ident: String;
                    const Map: Array of TIdentMapEntry) : Boolean
```

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Value` field matches `Int` and returns the corresponding identifier in `Ident`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: TIdentToInt (192), TintToIdent (192), IdentToInt (205), TIdentMapEntry (192)

2.4.25 InvalidPoint

Synopsis: Check whether a point is invalid.

Declaration:
```
function InvalidPoint(X: Integer;Y: Integer) : Boolean
function InvalidPoint(const At: TPoint) : Boolean
function InvalidPoint(const At: TSmallPoint) : Boolean
```

Visibility: default

Description: `InvalidPoint` returns `True` if the `X` and `Y` coordinates (of the `TPoint` or `TSmallPoint` records, if one of these versions is used) are -1.

See also: TPoint (194), TSmallPoint (196), PointsEqual (208)

2.4.26 LineStart

Synopsis: Finds the start of a line in `Buffer` before `BufPos`.

Declaration:
```
function LineStart(Buffer: PChar;BufPos: PChar) : PChar
```

Visibility: default

Description: `LineStart` reversely scans `Buffer` starting at `BufPos` for a linefeed character. It returns a pointer at the linefeed character.

2.4.27 NotifyGlobalLoading

Synopsis: Not yet implemented.

Declaration:
```
procedure NotifyGlobalLoading
```

Visibility: default

Description: Not yet implemented.

2.4.28 ObjectBinaryToText

Synopsis: Converts an object stream from a binary to a text format.

Declaration: `procedure ObjectBinaryToText(Input: TStream;Output: TStream)`

Visibility: default

Description: `ObjectBinaryToText` reads an object stream in binary format from `Input` and writes the object stream in text format to `Output`. No components are instantiated during the process, this is a pure conversion routine.

See also: ObjectTextToBinary (207)

2.4.29 ObjectResourceToText

Synopsis: Converts an object stream from a (windows) resource to a text format.

Declaration: `procedure ObjectResourceToText(Input: TStream;Output: TStream)`

Visibility: default

Description: `ObjectResourceToText` reads the resource header from the `Input` stream and then passes the streams to ObjectBinaryToText (207)

See also: ObjectBinaryToText (207), ObjectTextToResource (207)

2.4.30 ObjectTextToBinary

Synopsis: Converts an object stream from a text to a binary format.

Declaration: `procedure ObjectTextToBinary(Input: TStream;Output: TStream)`

Visibility: default

Description: Converts an object stream from a text to a binary format.

2.4.31 ObjectTextToResource

Synopsis: Converts an object stream from a text to a (windows) resource format.

Declaration: `procedure ObjectTextToResource(Input: TStream;Output: TStream)`

Visibility: default

Description: `ObjectTextToResource` reads an object stream in text format from `Input` and writes a resource stream to `Output`.

Note that for the current implementation of this method in Free Pascal, the output stream should support positioning. (e.g. it should not be a pipe)

See also: ObjectBinaryToText (207), ObjectResourceToText (207)

2.4.32 Point

Synopsis: Returns a `TPoint` record with the given coordinates.

Declaration:
```
function Point(AX: Integer;AY: Integer) : TPoint
```

Visibility: default

Description: `Point` returns a TPoint (194) record with the given coordinates `AX` and `AY` filled in.

See also: TPoint (194), SmallPoint (212), Rect (209), Bounds (200)

2.4.33 PointsEqual

Synopsis: Check whether two `TPoint` variables are equal.

Declaration:
```
function PointsEqual(const P1: TPoint;const P2: TPoint) : Boolean
function PointsEqual(const P1: TSmallPoint;const P2: TSmallPoint)
                        : Boolean
```

Visibility: default

Description: `PointsEqual` compares the `P1` and `P2` points (of type TPoint (194) or TSmallPoint (196)) and returns `True` if the `X` and `Y` coordinates of the points are equal, or `False` otherwise.

See also: TPoint (194), TSmallPoint (196), InvalidPoint (206)

2.4.34 ReadComponentRes

Synopsis: Read component properties from a resource in the current module

Declaration:
```
function ReadComponentRes(const ResName: String;Instance: TComponent)
                                  : TComponent
```

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `Nil`.

2.4.35 ReadComponentResEx

Synopsis: Read component properties from a resource in the specified module

Declaration:
```
function ReadComponentResEx(HInstance: THandle;const ResName: String)
                                    : TComponent
```

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `Nil`.

2.4.36 ReadComponentResFile

Synopsis: Read component properties from a specified resource file

Declaration:
```
function ReadComponentResFile(const FileName: String;
                                   Instance: TComponent) : TComponent
```

Visibility: default

Description: `ReadComponentResFile` starts reading properties for `Instance` from the file `FileName`. It creates a filestream from `FileName` and then calls the TStream.ReadComponentRes (335) method to read the state of the component from the stream.

See also: TStream.ReadComponentRes (335), WriteComponentResFile (214)

2.4.37 Rect

Synopsis: Returns a `TRect` record with the given coordinates.

Declaration:
```
function Rect(ALeft: Integer;ATop: Integer;ARight: Integer;
              ABottom: Integer) : TRect
```

Visibility: default

Description: `Rect` returns a TRect (195) record with the given top-left `(ALeft,ATop)` and bottom-right `(ABottom,ARight)` corners filled in.

No checking is done to see whether the coordinates are valid.

See also: TRect (195), Point (208), SmallPoint (212), Bounds (200)

2.4.38 RedirectFixupReferences

Synopsis: Redirects references under the `root` object from `OldRootName` to `NewRootName`

Declaration:
```
procedure RedirectFixupReferences(Root: TComponent;
                                  const OldRootName: String;
                                  const NewRootName: String)
```

Visibility: default

Description: `RedirectFixupReferences` examines the list of unresolved references and replaces references to a root object named `OldRootName` with references to root object `NewRootName`.

An application programmer should never need to call `RedirectFixupReferences`. This function can be used by an IDE to support redirection of broken component links.

See also: RemoveFixupReferences (212)

2.4.39 RegisterClass

Synopsis: Registers a class with the streaming system.

Declaration:
```
procedure RegisterClass(AClass: TPersistentClass)
```

Visibility: default

Description: `RegisterClass` registers the class `AClass` in the streaming system. After the class has been registered, it can be read from a stream when a reference to this class is encountered.

See also: RegisterClasses (210), RegisterClassAlias (210), RegisterComponents (210), UnregisterClass (213)

2.4.40 RegisterClassAlias

Synopsis: Registers a class alias with the streaming system.

Declaration:
```
procedure RegisterClassAlias(AClass: TPersistentClass;
                             const Alias: String)
```

Visibility: default

Description: `RegisterClassAlias` registers a class alias in the streaming system. If a reference to a class `Alias` is encountered in a stream, then an instance of the class `AClass` will be created instead by the streaming code.

See also: RegisterClass (209), RegisterClasses (210), RegisterComponents (210), UnregisterClass (213)

2.4.41 RegisterClasses

Synopsis: Registers multiple classes with the streaming system.

Declaration:
```
procedure RegisterClasses(AClasses: Array of TPersistentClass)
```

Visibility: default

Description: `RegisterClasses` registers the specified classes `AClass` in the streaming system. After the classes have been registered, they can be read from a stream when a reference to this class is encountered.

See also: RegisterClass (209), RegisterClassAlias (210), RegisterComponents (210), UnregisterClass (213)

2.4.42 RegisterComponents

Synopsis: Registers components for the component palette.

Declaration:
```
procedure RegisterComponents(const Page: String;
                             ComponentClasses: Array of TComponentClass)
```

Visibility: default

Description: `RegisterComponents` registers the component on the appropriate component page. The component pages can be used by an IDE to display the known components so an application programmer may pick and use the components in his programs.

Registercomponents inserts the component class in the correct component page. If the `RegisterComponentsProc` procedure is set, this is called as well. Note that this behaviour is different from Delphi's behaviour where an exception will be raised if the procedural variable is not set.

See also: RegisterClass (209), RegisterNoIcon (211)

2.4.43 RegisterFindGlobalComponentProc

Synopsis: Register a component searching handler

Declaration:
```
procedure RegisterFindGlobalComponentProc
                                        (AFindGlobalComponent: TFindGlobalComponent
```

Visibility: default

Description: `RegisterFindGlobalComponentProc` registers a global component search callback `AFindGlobalComponent`. When FindGlobalComponent (202) is called, then this callback will be used to search for the component.

Errors: None.

See also: FindGlobalComponent (202), UnRegisterFindGlobalComponentProc (213)

2.4.44 RegisterInitComponentHandler

Synopsis: Register a component initialization handler

Declaration:
```
procedure RegisterInitComponentHandler(ComponentClass: TComponentClass;
                                       Handler: TInitComponentHandler)
```

Visibility: default

Description: `RegisterInitComponentHandler` registers a component initialization handler `Handler` for the component `ComponentClass`. This handler will be used to initialize descendents of `ComponentClass` in the InitInheritedComponent (205) call.

See also: InitInheritedComponent (205), TInitComponentHandler (192)

2.4.45 RegisterIntegerConsts

Synopsis: Registers some integer-to-identifier mappings.

Declaration:
```
procedure RegisterIntegerConsts(IntegerType: Pointer;
                                IdentToIntFn: TIdentToInt;
                                IntToIdentFn: TIntToIdent)
```

Visibility: default

Description: `RegisterIntegerConsts` registers a pair of callbacks to be used when an integer of type `IntegerType` must be mapped to an identifier (using `IntToIdentFn`) or when an identifier must be mapper to an integer (using `IdentToIntFn`).

Component programmers can use `RegisterIntegerConsts` to associate a series of identifier strings with integer values for a property. A necessary condition is that the property should have a separate type declared using the `type integer` syntax. If a type of integer is defined in this way, an IDE can show symbolic names for the values of these properties.

The `IntegerType` should be a pointer to the type information of the integer type. The `IntToIdentFn` and `IdentToIntFn` are two callbacks that will be used when converting between the identifier and integer value and vice versa. The functions IdentToInt (205) and IntToIdent (206) can be used to implement these callback functions.

See also: TIdentToInt (192), TIntToIdent (192), IdentToInt (205), IntToIdent (206)

2.4.46 RegisterNoIcon

Synopsis: Registers components that have no icon on the component palette.

Declaration:
```
procedure RegisterNoIcon(ComponentClasses: Array of TComponentClass)
```

Visibility: default

Description: `RegisterNoIcon` performs the same function as RegisterComponents (210) except that it calls RegisterNoIconProc (199) instead of RegisterComponentsProc (199)

See also: RegisterNoIconProc (199), RegisterComponents (210)

2.4.47 RegisterNonActiveX

Synopsis: Register non-activex component.

Declaration:
```
procedure RegisterNonActiveX(ComponentClasses: Array of TComponentClass;
                             AxRegType: TActiveXRegType)
```

Visibility: default

Description: Not yet implemented in Free Pascal

2.4.48 RemoveFixupReferences

Synopsis: Removes references to rootname from the fixup list.

Declaration:
```
procedure RemoveFixupReferences(Root: TComponent;const RootName: String)
```

Visibility: default

Description: `RemoveFixupReferences` examines the list of unresolved references and removes references to a root object pointing at `Root` or a root component named `RootName`.

An application programmer should never need to call `RemoveFixupReferences`. This function can be used by an IDE to support removal of broken component links.

See also: RedirectFixupReferences (209)

2.4.49 RemoveFixups

Synopsis: Removes `Instance` from the fixup list.

Declaration:
```
procedure RemoveFixups(Instance: TPersistent)
```

Visibility: default

Description: `RemoveFixups` removes all entries for component `Instance` from the list of unresolved references.a

See also: RedirectFixupReferences (209), RemoveFixupReferences (212)

2.4.50 SmallPoint

Synopsis: Returns a `TSmallPoint` record with the given coordinates.

Declaration:
```
function SmallPoint(AX: SmallInt;AY: SmallInt) : TSmallPoint
```

Visibility: default

Description: `SmallPoint` returns a TSmallPoint (196) record with the given coordinates `AX` and `AY` filled in.

See also: TSmallPoint (196), Point (208), Rect (209), Bounds (200)

2.4.51 StartClassGroup

Synopsis: Start new class group.

Declaration: `procedure StartClassGroup(AClass: TPersistentClass)`

Visibility: default

Description: `StartClassGroup` starts a new class group and adds `AClass` to it.

The class registration and streaming mechanism allows to organize the classes in groups. This allows an IDE to form groups of classes, which can be enabled or disabled. It is not needed at Run-Time.

See also: GroupDescendentsWith (204), ActivateClassGroup (199), ClassGroupOf (201)

2.4.52 UnRegisterClass

Synopsis: Unregisters a class from the streaming system.

Declaration: `procedure UnRegisterClass(AClass: TPersistentClass)`

Visibility: default

Description: `UnregisterClass` removes the class `AClass` from the class definitions in the streaming system.

See also: UnRegisterClasses (213), UnRegisterModuleClasses (214), RegisterClass (209)

2.4.53 UnRegisterClasses

Synopsis: Unregisters multiple classes from the streaming system.

Declaration: `procedure UnRegisterClasses(AClasses: Array of TPersistentClass)`

Visibility: default

Description: `UnregisterClasses` removes the classes in `AClasses` from the class definitions in the streaming system.

2.4.54 UnregisterFindGlobalComponentProc

Synopsis: Remove a previously registered component searching handler.

Declaration: `procedure UnregisterFindGlobalComponentProc`
`(AFindGlobalComponent: TFindGlobalCompone`

Visibility: default

Description: `UnRegisterFindGlobalComponentProc` unregisters the previously registered global component search callback `AFindGlobalComponent`. After this call, when FindGlobalComponent (202) is called, then this callback will be no longer be used to search for the component.

Errors: None.

See also: FindGlobalComponent (202), RegisterFindGlobalComponentProc (210)

2.4.55 UnRegisterModuleClasses

Synopsis: Unregisters classes registered by module.

Declaration:
```
procedure UnRegisterModuleClasses(Module: HMODULE)
```

Visibility: default

Description: `UnRegisterModuleClasses` unregisters all classes which reside in the module `Module`. For each registered class, the definition pointer is checked to see whether it resides in the module, and if it does, the definition is removed.

See also: UnRegisterClass (213), UnRegisterClasses (213), RegisterClasses (210)

2.4.56 WriteComponentResFile

Synopsis: Write component properties to a specified resource file

Declaration:
```
procedure WriteComponentResFile(const FileName: String;
                                Instance: TComponent)
```

Visibility: default

Description: `WriteComponentResFile` starts writing properties of `Instance` to the file `FileName`. It creates a filestream from `FileName` and then calls TStream.WriteComponentRes (336) method to write the state of the component to the stream.

See also: TStream.WriteComponentRes (336), ReadComponentResFile (208)

2.5 EBitsError

2.5.1 Description

When an index of a bit in a TBits (254) is out of the valid range (`0` to `Count-1`) then a `EBitsError` exception is raised.

2.6 EClassNotFound

2.6.1 Description

When the streaming system needs to create a component, it looks for the class pointer (VMT) in the list of registered classes by its name. If this name is not found, then an `EClassNotFound` is raised.

2.7 EComponentError

2.7.1 Description

When an error occurs during the registration of a component, or when naming a component, then a `EComponentError` is raised. Possible causes are:

1. An name with an illegal character was assigned to a component.
2. A component with the same name and owner already exists.
3. The component registration system isn't set up properly.

2.8 EFCreateError

2.8.1 Description

When the operating system reports an error during creation of a new file in the Filestream Constructor (284), a `EFCreateError` is raised.

2.9 EFilerError

2.9.1 Description

This class serves as an ancestor class for exceptions that are raised when an error occurs during component streaming. A `EFilerError` exception is raised when a class is registered twice.

2.10 EFOpenError

2.10.1 Description

When the operating system reports an error during the opening of a file in the Filestream Constructor (284), a `EFOpenError` is raised.

2.11 EInvalidImage

2.11.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

2.12 EInvalidOperation

2.12.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibiliy purposes only.

2.13 EListError

2.13.1 Description

If an error occurs in one of the TList (300) or TStrings (349) methods, then a `EListError` exception is raised. This can occur in one of the following cases:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. An attempt was made to reduce the capacity of the list below the current element count.
4. An attempt was made to set the list count to a negative value.
5. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
6. An attempt was made to move an item to a position outside the list's bounds.

2.14 EMethodNotFound

2.14.1 Description

This exception is no longer used in the streaming system. This error is replaced by a EReadError (216).

2.15 EOutOfResources

2.15.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibiliy purposes only.

2.16 EParserError

2.16.1 Description

When an error occurs during the parsing of a stream, an `EParserError` is raised. Usually this indicates that an invalid token was found on the input stream, or the token read from the stream wasn't the expected token.

2.17 EReadError

2.17.1 Description

If an error occurs when reading from a stream, a `EReadError` exception is raised. Possible causes for this are:

1. Not enough data is available when reading from a stream
2. The stream containing a component's data contains invalid data. this will occurr only when reading a component from a stream.

2.18 EResNotFound

2.18.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

2.19 EStreamError

2.19.1 Description

An `EStreamError` is raised when an error occurs during reading from or writng to a stream: Possible causes are

1. Not enough data is available in the stream.
2. Trying to seek beyond the beginning or end of the stream.

3. Trying to set the capacity of a memory stream and no memory is available.
4. Trying to write to a read-only stream, such as a resource stream.
5. Trying to read from a write-only stream.

2.20 EStringListError

2.20.1 Description

When an error occurs in one of the methods of TStrings (349) then an `EStringListError` is raised. This can have one of the following causes:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
4. An attempt was made to add a duplicate entry to a TStringList (345) when TStringList.AllowDuplicates (345) is `False`.

2.21 EThread

2.21.1 Description

Thread error exception.

2.22 EThreadDestroyCalled

2.22.1 Description

Exception raised when a thread is destroyed illegally.

2.23 EWriteError

2.23.1 Description

If an error occurs when writing to a stream, a `EWriteError` exception is raised. Possible causes for this are:

1. The stream doesn't allow writing.
2. An error occurred when writing a property to a stream.

2.24 IDesignerNotify

2.24.1 Description

`IDesignerNotify` is an interface that can be used to communicate changes to a designer mechanism. It offers functionality for detecting changes, and notifications when the component is destroyed.

2.24.2 Method overview

Page	Property	Description
218	Modified	Notify that the component is modified.
218	Notification	Notification of owner changes

2.24.3 IDesignerNotify.Modified

Synopsis: Notify that the component is modified.

Declaration: `procedure Modified`

Visibility: default

Description: `Modified` can be used to notify a designer of changes, indicating that components should be streamed.

2.24.4 IDesignerNotify.Notification

Synopsis: Notification of owner changes

Declaration: `procedure Notification(AnObject: TPersistent;Operation: TOperation)`

Visibility: default

Description: `Notification` is the interface counterpart of TComponent.Notification (267) which is used to communicate adds to the components.

See also: TComponent.Notification (267)

2.25 IInterfaceComponentReference

2.25.1 Description

`IInterfaceComponentReference` is an interface to return the component that implements a given interface. It is implemented by TComponent (267).

2.25.2 Method overview

Page	Property	Description
218	GetComponent	Return component instance

2.25.3 IInterfaceComponentReference.GetComponent

Synopsis: Return component instance

Declaration: `function GetComponent : TComponent`

Visibility: default

Description: `GetComponent` returns the component instance.

Errors: None.

See also: TComponent (267)

2.26 IInterfaceList

2.26.1 Description

`IInterfaceList` is an interface for maintaining a list of interfaces, strongly resembling the standard TList (300) class. It offers the same list of public methods as `TList`, with the exception that it uses interfaces instead of pointers.

All interfaces in the list should descend from `IUnknown`.

More detailed descriptions of how the various methods behave can be found in the `TList` reference.

2.26.2 Method overview

Page	Property	Description
222	Add	Add an interface to the list
221	Clear	Clear the list
221	Delete	Remove an interface from the list
221	Exchange	Exchange 2 interfaces in the list
222	First	Return the first non-empty interface in the list.
219	Get	Retrieve an interface pointer from the list.
220	GetCapacity	Return the capacity of the list.
220	GetCount	Return the current number of elements in the list.
222	IndexOf	Return the index of an interface.
222	Insert	Insert an interface in the list.
222	Last	Returns the last non-nil interface in the list.
223	Lock	Lock the list
220	Put	Write an item to the list
223	Remove	Remove an interface from the list
220	SetCapacity	Set the capacity of the list
221	SetCount	Set the number of items in the list
223	Unlock	Unlock the list.

2.26.3 Property overview

Page	Property	Access	Description
223	Capacity	rw	Capacity of the list
224	Count	rw	Current number of elements in the list.
224	Items	rw	Provides Index-based, sequential, access to the interfaces in the list.

2.26.4 IInterfaceList.Get

Synopsis: Retrieve an interface pointer from the list.

Declaration:
```
function Get(i: Integer) : IUnknown
```

Visibility: default

Description: `Get` returns the interface pointer at position `i` in the list. It serves as the `Read` method for the Items (224) property.

See also: IInterfaceList.Items (224), TList.Items (306)

2.26.5 IInterfaceList.GetCapacity

Synopsis: Return the capacity of the list.

Declaration: `function GetCapacity : Integer`

Visibility: default

Description: `GetCapacity` returns the current capacity of the list. It serves as the `Read` method for the Capacity (223) property.

See also: IInterfaceList.Capacity (223), TList.Capacity (305)

2.26.6 IInterfaceList.GetCount

Synopsis: Return the current number of elements in the list.

Declaration: `function GetCount : Integer`

Visibility: default

Description: It serves as the `Read` method for the Count (224) property.

See also: IInterfaceList.Count (224), TList.Count (306)

2.26.7 IInterfaceList.Put

Synopsis: Write an item to the list

Declaration: `procedure Put(i: Integer;item: IUnknown)`

Visibility: default

Description: `Put` writes the interface `Item` at position `I` in the list. It servers as the `Write` method for the Items (224) property.

See also: IInterfaceList.Items (224), TList.Items (306)

2.26.8 IInterfaceList.SetCapacity

Synopsis: Set the capacity of the list

Declaration: `procedure SetCapacity(NewCapacity: Integer)`

Visibility: default

Description: `SetCapacity` sets the capacity of the list to `NewCapacity`. It serves as the `Write` method for the Capacity (223) property.

See also: IInterfaceList.Capacity (223), TList.Capacity (305)

2.26.9 IInterfaceList.SetCount

Synopsis: Set the number of items in the list

Declaration: `procedure SetCount(NewCount: Integer)`

Visibility: default

Description: `SetCount` sets the count of the list to `NewCount`. It serves as the `Write` method for the Capacity (223)

See also: IInterfaceList.Count (224), TList.Count (306)

2.26.10 IInterfaceList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear`

Visibility: default

Description: `Clear` removes all interfaces from the list. All interfaces in the list will be cleared (i.e. their reference count will decrease with 1)

See also: TList.Clear (301)

2.26.11 IInterfaceList.Delete

Synopsis: Remove an interface from the list

Declaration: `procedure Delete(index: Integer)`

Visibility: default

Description: `Delete` removes the interface at position `Index` from the list. It does this by explicitly clearing the interface and then removing the slot.

See also: TList.Clear (301), IInterfaceList.Add (222), IInterfaceList.Delete (221), IInterfaceList.Insert (222)

2.26.12 IInterfaceList.Exchange

Synopsis: Exchange 2 interfaces in the list

Declaration: `procedure Exchange(index1: Integer;index2: Integer)`

Visibility: default

Description: `Exchange` exchanges 2 interfaces in the list at locations `index1` and `Index2`.

See also: TList.Exchange (302), IInterfaceList.Add (222), IInterfaceList.Delete (221), IInterfaceList.Insert (222)

2.26.13 IInterfaceList.First

Synopsis: Return the first non-empty interface in the list.

Declaration: `function First : IUnknown`

Visibility: default

Description: `First` returns the first non-empty interface in the list.

See also: TList.First (303), IInterfaceList.IndexOf (222), IInterfaceList.Last (222)

2.26.14 IInterfaceList.IndexOf

Synopsis: Return the index of an interface.

Declaration: `function IndexOf(item: IUnknown) : Integer`

Visibility: default

Description: `IndexOf` returns the location in the list of the interface `Item`. If there is no such interface in the list, then -1 is returned.

See also: TList.IndexOf (303), IInterfaceList.First (222), IInterfaceList.Last (222)

2.26.15 IInterfaceList.Add

Synopsis: Add an interface to the list

Declaration: `function Add(item: IUnknown) : Integer`

Visibility: default

Description: `Add` adds the interface `Item` to the list, and returns the position at which it has been added.

See also: TList.Add (301), IInterfaceList.Insert (222), IInterfaceList.Delete (221)

2.26.16 IInterfaceList.Insert

Synopsis: Insert an interface in the list.

Declaration: `procedure Insert(i: Integer;item: IUnknown)`

Visibility: default

Description: `Insert` inserts the interface `Item` in the list, at position `I`, shifting all items one position.

See also: TList.Insert (303), IInterfaceList.Add (222), IInterfaceList.Delete (221)

2.26.17 IInterfaceList.Last

Synopsis: Returns the last non-nil interface in the list.

Declaration: `function Last : IUnknown`

Visibility: default

Description: `Last` returns the last non-empty interface in the list.

See also: TList.Last (304), IInterfaceList.First (222), IInterfaceList.IndexOf (222)

2.26.18 IInterfaceList.Remove

Synopsis: Remove an interface from the list

Declaration: `function Remove(item: IUnknown) : Integer`

Visibility: default

Description: `Remove` searches for the first occurrence of `Item` in the list and deletes it.

See also: TList.Remove (304), IInterfaceList.Delete (221), IInterfaceList.IndexOf (222)

2.26.19 IInterfaceList.Lock

Synopsis: Lock the list

Declaration: `procedure Lock`

Visibility: default

Description: `Lock` locks the list and returns an instance to the internal list. After a call to lock, the object list can only be accessed by the current thread, untill UnLock (223) is called.

See also: TList.Lock (300), IInterfaceList.Unlock (223)

2.26.20 IInterfaceList.Unlock

Synopsis: Unlock the list.

Declaration: `procedure Unlock`

Visibility: default

Description: `Unlock` unlocks a locked list. After a call to `UnLock`, other lists are again able to access the list.

See also: TList.UnLock (300), IInterfaceList.Lock (223)

2.26.21 IInterfaceList.Capacity

Synopsis: Capacity of the list

Declaration: `Property Capacity : Integer`

Visibility: default

Access: Read,Write

Description: `Capacity` is the maximum number of elements the list can hold without needing to reallocate memory for the list. It can be set to improve speed when adding a lot of items to the list.

See also: TList.Capacity (305), IInterfaceList.Count (224)

2.26.22 IInterfaceList.Count

Synopsis: Current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: default

Access: Read,Write

Description: `Count` is the current number of elements in the list. Setting it to a larger number will allocate empty slots. Setting it to a smaller number will clear any interfaces that fall outside the new border.

See also: IInterfaceList.Capacity (223), TList.Count (306)

2.26.23 IInterfaceList.Items

Synopsis: Provides Index-based, sequential, access to the interfaces in the list.

Declaration: `Property Items[index: Integer]: IUnknown; default`

Visibility: default

Access: Read,Write

Description: `Items` is the default property of the interface list and provides index-based array access to the interfaces in the list. Allowed values for `Index` include 0 to `Count-1`

See also: IInterfaceList.Count (224), TList.Items (306)

2.27 IStreamPersist

2.27.1 Description

`IStreamPersist` defines an interface for object persistence streaming to a stream. Any class implementing this interface is expected to be able to save or load it's state from or to a stream.

2.27.2 Method overview

Page	Property	Description
224	LoadFromStream	Load persistent data from stream.
225	SaveToStream	Save persistent data to stream.

2.27.3 IStreamPersist.LoadFromStream

Synopsis: Load persistent data from stream.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: default

Description: `LoadFromStream` is the method called when the object should load it's state from the stream `stream`. It should be able to read the data which was written using the `SavetoStream` method.

See also: TPersistent (317), TComponent (267), TStream (332), IStreamPersist.SaveToStream (225)

2.27.4 IStreamPersist.SaveToStream

Synopsis: Save persistent data to stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: default

Description: `SaveFromStream` is the method called when the object should load it's state from the stream `stream`. The data written by this method should be readable by the `LoadFromStream` method.

See also: TPersistent (317), TComponent (267), TStream (332), IStreamPersist.LoadFromStream (224)

2.28 IStringsAdapter

2.28.1 Description

Is not yet supported in Free Pascal.

2.28.2 Method overview

Page	Property	Description
225	ReferenceStrings	Add a reference to the indicated strings.
225	ReleaseStrings	Release the reference to the strings.

2.28.3 IStringsAdapter.ReferenceStrings

Synopsis: Add a reference to the indicated strings.

Declaration: `procedure ReferenceStrings(S: TStrings)`

Visibility: default

2.28.4 IStringsAdapter.ReleaseStrings

Synopsis: Release the reference to the strings.

Declaration: `procedure ReleaseStrings`

Visibility: default

2.29 TAbstractObjectReader

2.29.1 Description

The Free Pascal streaming mechanism, while compatible with Delphi's mechanism, differs from it in the sense that the streaming mechanism uses a driver class when streaming components. The `TAbstractObjectReader` class is the base driver class for reading property values from streams. It consists entirely of abstract methods, which must be implemented by descendent classes.

Different streaming mechanisms can be implemented by making a descendent from `TAbstractObjectReader`. The TBinaryObjectReader (243) class is such a descendent class, which streams data in binary (Delphi compatible) format.

All methods described in this class, mustbe implemented by descendent classes.

2.29.2 Method overview

Page	Property	Description
227	BeginComponent	Marks the reading of a new component.
227	BeginProperty	Marks the reading of a property value.
227	BeginRootComponent	Starts the reading of the root component.
226	NextValue	Returns the type of the next value in the stream.
227	Read	Read raw data from stream
228	ReadBinary	Read binary data from the stream.
229	ReadCurrency	Read a currency value from the stream.
229	ReadDate	Read a date value from the stream.
228	ReadFloat	Read a float value from the stream.
229	ReadIdent	Read an identifier from the stream.
230	ReadInt16	Read a 16-bit integer from the stream.
230	ReadInt32	Read a 32-bit integer from the stream.
231	ReadInt64	Read a 64-bit integer from the stream.
230	ReadInt8	Read an 8-bit integer from the stream.
231	ReadSet	Reads a set from the stream.
228	ReadSingle	Read a single (real-type) value from the stream.
231	ReadStr	Read a shortstring from the stream
232	ReadString	Read a string of type `StringType` from the stream.
226	ReadValue	Reads the type of the next value.
232	ReadWideString	Read a widestring value from the stream.
232	SkipComponent	Skip till the end of the component.
233	SkipValue	Skip the current value.

2.29.3 TAbstractObjectReader.NextValue

Synopsis: Returns the type of the next value in the stream.

Declaration: `function NextValue : TValueType; Virtual; Abstract`

Visibility: public

Description: This function should return the type of the next value in the stream, but should not read it, i.e. the stream position should not be altered by this method. This is used to 'peek' in the stream what value is next.

See also: TAbstractObjectReader.ReadValue (226)

2.29.4 TAbstractObjectReader.ReadValue

Synopsis: Reads the type of the next value.

Declaration: `function ReadValue : TValueType; Virtual; Abstract`

Visibility: public

Description: This function returns the type of the next value in the stream and reads it. i.e. after the call to this method, the stream is positioned to read the value of the type returned by this function.

See also: TAbstractObjectReader.ReadValue (226)

2.29.5 TAbstractObjectReader.BeginRootComponent

Synopsis: Starts the reading of the root component.

Declaration:
```
procedure BeginRootComponent;  Virtual;  Abstract
```

Visibility: public

Description: This function can be used to initialize the driver class for reading a component. It is called once at the beginning of the read process, and is immediatly followed by a call to BeginComponent (227).

See also: TAbstractObjectReader.BeginComponent (227)

2.29.6 TAbstractObjectReader.BeginComponent

Synopsis: Marks the reading of a new component.

Declaration:
```
procedure BeginComponent(var Flags: TFilerFlags;var AChildPos: Integer;
                         var CompClassName: String;var CompName: String)
                         ;  Virtual;  Abstract
```

Visibility: public

Description: This method is called when the streaming process wants to start reading a new component.

Descendent classes should override this method to read the start of a component new component definition and return the needed arguments. `Flags` should be filled with any flags that were found at the component definition, as well as `AChildPos`. The `CompClassName` should be filled with the class name of the streamed component, and the `CompName` argument should be filled with the name of the component.

See also: TAbstractObjectReader.BeginRootComponent (227), TAbstractObjectReader.BeginProperty (227)

2.29.7 TAbstractObjectReader.BeginProperty

Synopsis: Marks the reading of a property value.

Declaration:
```
function BeginProperty : String;  Virtual;  Abstract
```

Visibility: public

Description: `BeginProperty` is called by the streaming system when it wants to read a new property. The return value of the function is the name of the property which can be read from the stream.

See also: TAbstractObjectReader.BeginComponent (227)

2.29.8 TAbstractObjectReader.Read

Synopsis: Read raw data from stream

Declaration:
```
procedure Read(var Buf;Count: LongInt);  Virtual;  Abstract
```

Visibility: public

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in production code as it will totally mess up the streaming.

See also: TBinaryObjectReader.Read (246), TReader.Read (322)

2.29.9 TAbstractObjectReader.ReadBinary

Synopsis: Read binary data from the stream.

Declaration: `procedure ReadBinary(const DestData: TMemoryStream); Virtual; Abstract`

Visibility: public

Description: `ReadBinary` is called when binary data should be read from the stream (i.e. after ReadValue (226) returned a valuetype of `vaBinary`). The data should be stored in the `DestData` memory stream by descendent classes.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.10 TAbstractObjectReader.ReadFloat

Synopsis: Read a float value from the stream.

Declaration: `function ReadFloat : Extended; Virtual; Abstract`

Visibility: public

Description: `ReadFloat` is called by the streaming system when it wants to read a float from the stream (i.e. after ReadValue (226) returned a valuetype of `vaExtended`). The return value should be the value of the float.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.11 TAbstractObjectReader.ReadSingle

Synopsis: Read a single (real-type) value from the stream.

Declaration: `function ReadSingle : Single; Virtual; Abstract`

Visibility: public

Description: `ReadSingle` is called by the streaming system when it wants to read a single-type float from the stream (i.e. after ReadValue (226) returned a valuetype of `vaSingle`). The return value should be the value of the float.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.12 TAbstractObjectReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency; Virtual; Abstract`

Visibility: public

Description: `ReadCurrency` is called when a currency-typed value should be read from the stream. This abstract method should be overidden by descendent classes, and should return the currency value read from the stream.

See also: TAbstractObjectWriter.WriteCurrency (236)

2.29.13 TAbstractObjectReader.ReadDate

Synopsis: Read a date value from the stream.

Declaration: `function ReadDate : TDateTime; Virtual; Abstract`

Visibility: public

Description: `ReadDate` is called by the streaming system when it wants to read a date/time value from the stream (i.e. after ReadValue (226) returned a valuetype of `vaDate`). The return value should be the date/time value. (This value can be stored as a float, since `TDateTime` is nothing but a float.)

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.14 TAbstractObjectReader.ReadIdent

Synopsis: Read an identifier from the stream.

Declaration: `function ReadIdent(ValueType: TValueType) : String; Virtual; Abstract`

Visibility: public

Description: `ReadIdent` is called by the streaming system if it expects to read an identifier of type `ValueType` from the stream after a call to ReadValue (226) returned `vaIdent`. The identifier should be returned as a string. Note that in some cases the identifier does not actually have to be in the stream;

Table 2.16:

ValueType	Expected value
vaIdent	Read from stream.
vaNil	'Nil'. This does not have to be read from the stream.
vaFalse	'False'. This does not have to be read from the stream.
vaTrue	'True'. This does not have to be read from the stream.
vaNull	'Null'. This does not have to be read from the stream.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.15 TAbstractObjectReader.ReadInt8

Synopsis: Read an 8-bit integer from the stream.

Declaration: `function ReadInt8 : ShortInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt8` is called by the streaming process if it expects to read an integer value with a size of 8 bits (1 byte) from the stream (i.e. after ReadValue (226) returned a valuetype of `vaInt8`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 1 byte.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.16 TAbstractObjectReader.ReadInt16

Synopsis: Read a 16-bit integer from the stream.

Declaration: `function ReadInt16 : SmallInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt16` is called by the streaming process if it expects to read an integer value with a size of 16 bits (2 bytes) from the stream (i.e. after ReadValue (226) returned a valuetype of `vaInt16`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 2 bytes.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.17 TAbstractObjectReader.ReadInt32

Synopsis: Read a 32-bit integer from the stream.

Declaration: `function ReadInt32 : LongInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt32` is called by the streaming process if it expects to read an integer value with a size of 32 bits (4 bytes) from the stream (i.e. after ReadValue (226) returned a valuetype of `vaInt32`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 4 bytes.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.18 TAbstractObjectReader.ReadInt64

Synopsis: Read a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64; Virtual; Abstract`

Visibility: public

Description: `ReadInt64` is called by the streaming process if it expects to read an int64 value with a size of 64 bits (8 bytes) from the stream (i.e. after ReadValue (226) returned a valuetype of `vaInt64`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 8 bytes.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.19 TAbstractObjectReader.ReadSet

Synopsis: Reads a set from the stream.

Declaration: `function ReadSet(EnumType: Pointer) : Integer; Virtual; Abstract`

Visibility: public

Description: This method is called by the streaming system if it expects to read a set from the stream (i.e. after ReadValue (226) returned a valuetype of `vaSet`). The return value is the contents of the set, encoded in a bitmask the following way:

For each (enumerated) value in the set, the bit corresponding to the ordinal value of the enumerated value should be set. i.e. as `1 shl ord(value)`.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TabstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadStr (231), TabstractObjectReader.ReadString (232)

2.29.20 TAbstractObjectReader.ReadStr

Synopsis: Read a shortstring from the stream

Declaration: `function ReadStr : String; Virtual; Abstract`

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a shortstring from the stream (i.e. after ReadValue (226) returned a valuetype of `vaLString`,`vaWstring` or `vaString`). The return value is the string.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadString (232)

2.29.21 TAbstractObjectReader.ReadString

Synopsis: Read a string of type `StringType` from the stream.

Declaration:
```
function ReadString(StringType: TValueType) : String;  Virtual
                          ;  Abstract
```

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a string from the stream (i.e. after ReadValue (226) returned a valuetype of `vaLString`,`vaWstring` or `vaString`). The return value is the string.

See also: TAbstractObjectReader.ReadFloat (228), TAbstractObjectReader.ReadDate (229), TAbstractObjectReader.ReadSingle (228), TAbstractObjectReader.ReadIdent (229), TAbstractObjectReader.ReadInt8 (230), TAbstractObjectReader.ReadInt16 (230), TAbstractObjectReader.ReadInt32 (230), TAbstractObjectReader.ReadInt64 (231), TabstractObjectReader.ReadSet (231), TabstractObjectReader.ReadStr (231)

2.29.22 TAbstractObjectReader.ReadWideString

Synopsis: Read a widestring value from the stream.

Declaration:
```
function ReadWideString : WideString;  Virtual;  Abstract
```

Visibility: public

Description: `ReadWideString` is called when a widestring-typed value should be read from the stream. This abstract method should be overidden by descendent classes.

See also: TAbstractObjectWriter.WriteWideString (237)

2.29.23 TAbstractObjectReader.SkipComponent

Synopsis: Skip till the end of the component.

Declaration:
```
procedure SkipComponent(SkipComponentInfos: Boolean);  Virtual
                              ;  Abstract
```

Visibility: public

Description: This method is used to skip the entire declaration of a component in the stream. Each descendent of `TAbstractObjectReader` should implement this in a way which is optimal for the implemented stream format.

See also: TAbstractObjectReader.BeginComponent (227), TAbstractObjectReader.SkipValue (233)

2.29.24 TAbstractObjectReader.SkipValue

Synopsis: Skip the current value.

Declaration: `procedure SkipValue; Virtual; Abstract`

Visibility: public

Description: `SkipValue` should be used when skipping a value in the stream; The method should determine the type of the value which should be skipped by itself, if this is necessary.

See also: TAbstractObjectReader.SkipComponent (232)

2.30 TAbstractObjectWriter

2.30.1 Description

Abstract driver class for writing component data.

2.30.2 Method overview

Page	Property	Description
233	BeginCollection	Start writing a collection.
234	BeginComponent	Start writing a component
234	BeginList	Start writing a list.
234	BeginProperty	Start writing a property
234	EndList	Mark the end of a list.
234	EndProperty	Marks the end of writing of a property.
235	Write	Write raw data to stream
235	WriteBinary	Writes binary data to the stream.
235	WriteBoolean	Writes a boolean value to the stream.
236	WriteCurrency	Write a currency value to the stream
236	WriteDate	Writes a date type to the stream.
235	WriteFloat	Writes a float value to the stream.
236	WriteIdent	Writes an identifier to the stream.
236	WriteInteger	Writes an integer value to the stream
236	WriteMethodName	Writes a methodname to the stream.
237	WriteSet	Writes a set value to the stream.
235	WriteSingle	Writes a single-type real value to the stream.
237	WriteString	Writes a string value to the stream.
237	WriteWideString	Write a widestring value to the stream

2.30.3 TAbstractObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection; Virtual; Abstract`

Visibility: public

Description: Start writing a collection.

2.30.4 TAbstractObjectWriter.BeginComponent

Synopsis: Start writing a component

Declaration:
```
procedure BeginComponent(Component: TComponent;Flags: TFilerFlags;
                         ChildPos: Integer);  Virtual;  Abstract
```

Visibility: public

Description: Start writing a component

2.30.5 TAbstractObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration:
```
procedure BeginList;  Virtual;  Abstract
```

Visibility: public

Description: Start writing a list.

2.30.6 TAbstractObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration:
```
procedure EndList;  Virtual;  Abstract
```

Visibility: public

Description: Mark the end of a list.

2.30.7 TAbstractObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration:
```
procedure BeginProperty(const PropName: String);  Virtual;  Abstract
```

Visibility: public

Description: Start writing a property

2.30.8 TAbstractObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration:
```
procedure EndProperty;  Virtual;  Abstract
```

Visibility: public

Description: Marks the end of writing of a property.

2.30.9 TAbstractObjectWriter.Write

Synopsis: Write raw data to stream

Declaration: `procedure Write(const Buffer;Count: LongInt); Virtual; Abstract`

Visibility: public

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TBinaryObjectWriter.Write (251), TWriter.Write (373)

2.30.10 TAbstractObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer;Count: LongInt); Virtual; Abstract`

Visibility: public

Description: Writes binary data to the stream.

2.30.11 TAbstractObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean); Virtual; Abstract`

Visibility: public

Description: Writes a boolean value to the stream.

2.30.12 TAbstractObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended); Virtual; Abstract`

Visibility: public

Description: Writes a float value to the stream.

2.30.13 TAbstractObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single); Virtual; Abstract`

Visibility: public

Description: Writes a single-type real value to the stream.

2.30.14 TAbstractObjectWriter.WriteCurrency

Synopsis: Write a currency value to the stream

Declaration: `procedure WriteCurrency(const Value: Currency); Virtual; Abstract`

Visibility: public

Description: `WriteCurrency` is called when a currency-typed value should be written to the stream. This abstract method should be overidden by descendent classes.

See also: TAbstractObjectReader.ReadCurrency (229)

2.30.15 TAbstractObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime); Virtual; Abstract`

Visibility: public

Description: Writes a date type to the stream.

2.30.16 TAbstractObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String); Virtual; Abstract`

Visibility: public

Description: Writes an identifier to the stream.

2.30.17 TAbstractObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream

Declaration: `procedure WriteInteger(Value: Int64); Virtual; Abstract`

Visibility: public

Description: Writes an integer value to the stream

2.30.18 TAbstractObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: String); Virtual; Abstract`

Visibility: public

Description: Writes a methodname to the stream.

2.30.19 TAbstractObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt;SetType: Pointer); Virtual; Abstract`

Visibility: public

Description: Writes a set value to the stream.

2.30.20 TAbstractObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: String); Virtual; Abstract`

Visibility: public

Description: Writes a string value to the stream.

2.30.21 TAbstractObjectWriter.WriteWideString

Synopsis: Write a widestring value to the stream

Declaration: `procedure WriteWideString(const Value: WideString); Virtual; Abstract`

Visibility: public

Description: `WriteCurrency` is called when a currency-typed value should be written to the stream. This abstract method should be overidden by descendent classes.

See also: TAbstractObjectReader.ReadWideString (232)

2.31 TBasicAction

2.31.1 Description

`TBasicAction` implements a basic action class from which all actions are derived. It introduces all basic methods of an action, and implements functionality to maintain a list of clients, i.e. components that are connected with this action.

Do not create instances of `TBasicAction`. Instead, create a descendent class and create an instance of this class instead.

2.31.2 Method overview

Page	Property	Description
238	Create	Creates a new instance of a TBasicAction (237) class.
238	Destroy	Destroys the action.
239	Execute	Triggers the OnExecute (240) event
239	ExecuteTarget	Executes the action on the `Target` object
238	HandlesTarget	Determines whether `Target` can be handled by this action
239	RegisterChanges	Registers a new client with the action.
240	UnRegisterChanges	Unregisters a client from the list of clients
240	Update	Triggers the OnUpdate (241) event
239	UpdateTarget	Notify client controls when the action updates itself.

2.31.3 Property overview

Page	Property	Access	Description
240	ActionComponent	rw	Returns the component that initiated the action.
240	OnExecute	rw	Event triggered when the action executes.
241	OnUpdate	rw	Event trigged when the application is idle.

2.31.4 TBasicAction.Create

Synopsis: Creates a new instance of a TBasicAction (237) class.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` calls the inherited constructor, and then initializes the list of clients controls (or action lists) by adding the `AClient` argument to the list of client controls.

Under normal circumstances it should not be necessary to create a `TBasicAction` descendent manually, actions are created in an IDE.

See also: TBasicAction.Destroy (238), TBasicAction.AssignClient (237)

2.31.5 TBasicAction.Destroy

Synopsis: Destroys the action.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the list of client controls and then calls the inherited destructor.

An application programmer should not call `Destroy` directly; Instead `Free` should be called, if it needs to be called at all. Normally the controlling class (e.g. a TActionList) will destroy the action.

2.31.6 TBasicAction.HandlesTarget

Synopsis: Determines whether `Target` can be handled by this action

Declaration: `function HandlesTarget(Target: TObject) : Boolean;  Virtual`

Visibility: public

Description: `HandlesTarget` returns `True` if `Target` is a valid client for this action and if so, if it is in a suitable state to execute the action. An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

In `TBasicAction` this method is empty; descendent classes should override this method to implement appropriate checks.

See also: TBasicAction.UpdateTarget (239), TBasicAction.ExecuteTarget (239)

2.31.7 TBasicAction.UpdateTarget

Synopsis: Notify client controls when the action updates itself.

Declaration: `procedure UpdateTarget(Target: TObject); Virtual`

Visibility: public

Description: `UpdateTarget` should update the client control specified by `Target` when the action updates itself. In `TBasicAction`, the implementation of `UpdateTarget` is empty. Descendent classes should override and implement `UpdateTarget` to actually update the `Target` object.

An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

See also: TBasicAction.HandlesTarget (238), TBasicAction.ExecuteTarget (239)

2.31.8 TBasicAction.ExecuteTarget

Synopsis: Executes the action on the `Target` object

Declaration: `procedure ExecuteTarget(Target: TObject); Virtual`

Visibility: public

Description: `ExecuteTarget` performs the action on the `Target` object. In `TBasicAction` this method does nothing. Descendent classes should implement the action to be performed. For instance an action to post data in a dataset could call the `Post` method of the dataset.

An application programmer should never call `ExecuteTarget` directly.

See also: TBasicAction.HandlesTarget (238), TBasicAction.ExecuteTarget (239), TBasicAction.Execute (239)

2.31.9 TBasicAction.Execute

Synopsis: Triggers the OnExecute (240) event

Declaration: `function Execute : Boolean; Dynamic`

Visibility: public

Description: `Execute` triggers the `OnExecute` event, if one is assigned. It returns `True` if the event handler was called, `False` otherwise.

2.31.10 TBasicAction.RegisterChanges

Synopsis: Registers a new client with the action.

Declaration: `procedure RegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `RegisterChanges` adds `Value` to the list of clients.

See also: TBasicAction.UnregisterChanges (240)

2.31.11 TBasicAction.UnRegisterChanges

Synopsis: Unregisters a client from the list of clients

Declaration: `procedure UnRegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `UnregisterChanges` removes `Value` from the list of clients. This is called for instance when the action is destroyed, or when the client is assigned a new action.

See also: TBasicAction.UnregisterChanges (240), TBasicAction.Destroy (238)

2.31.12 TBasicAction.Update

Synopsis: Triggers the OnUpdate (241) event

Declaration: `function Update : Boolean;  Virtual`

Visibility: public

Description: `Update` triggers the `OnUpdate` event, if one is assigned. It returns `True` if the event was triggered, or `False` if no event was assigned.

Application programmers should never run `Update` directly. The `Update` method is called automatically by the action mechanism; Normally this is in the Idle time of an application. An application programmer should assign the OnUpdate (241) event, and perform any checks in that handler.

See also: TBasicAction.OnUpdate (241), TBasicAction.Execute (239), TBAsicAction.UpdateTarget (239)

2.31.13 TBasicAction.ActionComponent

Synopsis: Returns the component that initiated the action.

Declaration: `Property ActionComponent : TComponent`

Visibility: public

Access: Read,Write

Description: `ActionComponent` is set to the component that caused the action to execute, e.g. a toolbutton or a menu item. The property is set just before the action executes, and is reset to nil after the action was executed.

See also: TBasicAction.Execute (239), TBasicAction.OnExecute (240)

2.31.14 TBasicAction.OnExecute

Synopsis: Event triggered when the action executes.

Declaration: `Property OnExecute : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnExecute` is the event triggered when the action is activated (executed). The event is triggered e.g. when the user clicks e.g. on a menu item or a button associated to the action. The application programmer should provide a `OnExecute` event handler to execute whatever code is necessary when the button is pressed or the menu item is chosen.

Note that assigning an `OnExecute` handler will result in the Execute (239) method returning a `True` value. Predefined actions (such as dataset actions) will check the result of `Execute` and will not perform their normal task if the `OnExecute` handler was called.

See also: TBasicAction.Execute (239), TBasicAction.OnUpdate (241)

2.31.15 TBasicAction.OnUpdate

Synopsis: Event trigged when the application is idle.

Declaration: `Property OnUpdate : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnUpdate` is the event triggered when the application is idle, and the action is being updated. The `OnUpdate` event can be used to set the state of the action, for instance disable it if the action cannot be executed at this point in time.

See also: TBasicAction.Update (240), TBasicAction.OnExecute (240)

2.32 TBasicActionLink

2.32.1 Description

`TBasicActionLink` links an Action to its clients. With each client for an action, a `TBasicActionLink` class is instantiated to handle the communication between the action and the client. It passes events between the action and its clients, and thus presents the action with a uniform interface to the clients.

An application programmer should never use a `TBasicActionLink` instance directly; They are created automatically when an action is associated with a component. Component programmers should create specialized descendents of `TBasicActionLink` which communicate changes in the action to the component.

2.32.2 Method overview

Page	Property	Description
242	Create	Creates a new instance of the TBasicActionLink class
242	Destroy	Destroys the TBasicActionLink instance.
242	Execute	Calls the action's Execute method.
243	Update	Calls the action's Update method

2.32.3 Property overview

Page	Property	Access	Description
243	Action	rw	The action to which the link was assigned.
243	OnChange	rw	Event handler triggered when the action's properties change

2.32.4 TBasicActionLink.Create

Synopsis: Creates a new instance of the TBasicActionLink class

Declaration: `constructor Create(AClient: TObject); Virtual`

Visibility: public

Description: `Create` creates a new instance of a `TBasicActionLink` and assigns `AClient` as the client of the link.

Application programmers should never instantiate `TBasicActionLink` classes directly. An instance is created automatically when an action is assigned to a control (client).

Component programmers can override the create constructor to initialize further properties.

See also: TBasicActionLink.Destroy (242)

2.32.5 TBasicActionLink.Destroy

Synopsis: Destroys the TBasicActionLink instance.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` unregisters the `TBasicActionLink` with the action, and then calls the inherited destructor.

Application programmers should never call `Destroy` directly. If a link should be destroyed at all, the `Free` method should be called instead.

See also: TBasicActionLink.Create (242)

2.32.6 TBasicActionLink.Execute

Synopsis: Calls the action's Execute method.

Declaration: `function Execute(AComponent: TComponent) : Boolean; Virtual`

Visibility: public

Description: `Execute` sets the ActionComponent (240) property of the associated Action (243) to `AComponent` and then calls the Action's execute (239) method. After the action has executed, the `ActionComponent` property is cleared again.

The return value of the function is the return value of the Action's execute method.

Application programmers should never call `Execute` directly. This method will be called automatically when the associated control is activated. (e.g. a button is clicked on)

Component programmers should call `Execute` whenever the action should be activated.

See also: TBasicActionLink.Action (243), TBasicAction.ActionComponent (240), TBasicAction.Execute (239), TBasicAction.onExecute (240)

2.32.7 TBasicActionLink.Update

Synopsis: Calls the action's Update method

Declaration: `function Update : Boolean; Virtual`

Visibility: public

Description: `Update` calls the associated Action's Update (240) methoda.

Component programmers can override the `Update` method to provide additional processing when the `Update` method occurs.

2.32.8 TBasicActionLink.Action

Synopsis: The action to which the link was assigned.

Declaration: `Property Action : TBasicAction`

Visibility: public

Access: Read,Write

Description: `Action` represents the Action (237) which was assigned to the client. Setting this property will unregister the client at the old action (if one existed) and registers the client at the new action.

See also: TBasicAction (237)

2.32.9 TBasicActionLink.OnChange

Synopsis: Event handler triggered when the action's properties change

Declaration: `Property OnChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChange` is the event triggered when the action's properties change.

Application programmers should never need to assign this event. Component programmers can assign this event to have a client control reflect any changes in an Action's properties.

See also: TBasicActionLink.Change (241), TBasicAction.Change (237)

2.33 TBinaryObjectReader

2.33.1 Description

The `TBinaryObjectReader` class reads component data stored in binary form in a file. For this, it overrides or implements all abstract methods from TAbstractObjectReader (225). No new functionality is added by this class, it is a driver class for the streaming system.

2.33.2 Method overview

Page	Property	Description
245	BeginComponent	Start reading a component.
245	BeginProperty	Start reading a property.
245	BeginRootComponent	Start reading the root component.
244	Create	Creates a new binary data reader instance.
244	Destroy	Destroys the binary data reader.
245	NextValue	Return the type of the next value.
246	Read	Read raw data from stream
246	ReadBinary	Start reading a binary value.
246	ReadCurrency	Read a currency value from the stream.
247	ReadDate	Read a date.
246	ReadFloat	Read a float value
247	ReadIdent	Read an identifier
247	ReadInt16	Read a 16-bits integer.
248	ReadInt32	Read a 32-bits integer.
248	ReadInt64	Read a 64-bits integer.
247	ReadInt8	Read an 8-bits integer.
248	ReadSet	Read a set
246	ReadSingle	Read a single-size float value
248	ReadStr	Read a short string
248	ReadString	Read a string
245	ReadValue	Read the next value in the stream
249	ReadWideString	Read a widestring value from the stream.
249	SkipComponent	Skip a component's data
249	SkipValue	Skip a value's data

2.33.3 TBinaryObjectReader.Create

Synopsis: Creates a new binary data reader instance.

Declaration: `constructor Create(Stream: TStream;BufSize: Integer)`

Visibility: public

Description: `Create` instantiates a new binary component data reader. The `Stream` stream is the stream from which data will be read. The `BufSize` argument is the size of the internal buffer that will be used by the reader. This can be used to optimize the reading process.

See also: TAbstractObjectReader (225)

2.33.4 TBinaryObjectReader.Destroy

Synopsis: Destroys the binary data reader.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the buffer allocated when the instance was created. It also positions the stream on the last used position in the stream (the buffering may cause the reader to read more bytes than were actually used.)

See also: TBinaryObjectReader.Create (244)

2.33.5 TBinaryObjectReader.NextValue

Synopsis: Return the type of the next value.

Declaration:
```
function NextValue : TValueType;  Override
```

Visibility: public

Description: `NextValue` returns the type of the next value in a binary stream, but does not read the value.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.6 TBinaryObjectReader.ReadValue

Synopsis: Read the next value in the stream

Declaration:
```
function ReadValue : TValueType;  Override
```

Visibility: public

Description: `NextValue` reads the next value in a binary stream and returns the type of the read value.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.7 TBinaryObjectReader.BeginRootComponent

Synopsis: Start reading the root component.

Declaration:
```
procedure BeginRootComponent;  Override
```

Visibility: public

Description: `BeginRootComponent` starts reading the root component in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.8 TBinaryObjectReader.BeginComponent

Synopsis: Start reading a component.

Declaration:
```
procedure BeginComponent(var Flags: TFilerFlags;var AChildPos: Integer;
                         var CompClassName: String;var CompName: String)
                         ;  Override
```

Visibility: public

Description: This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.9 TBinaryObjectReader.BeginProperty

Synopsis: Start reading a property.

Declaration:
```
function BeginProperty : String;  Override
```

Visibility: public

Description: This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.10 TBinaryObjectReader.Read

Synopsis: Read raw data from stream

Declaration: `procedure Read(var Buf;Count: LongInt);  Override`

Visibility: public

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in production code as it will totally mess up the streaming.

See also: TAbstractObjectReader.Read (227), TReader.Read (322)

2.33.11 TBinaryObjectReader.ReadBinary

Synopsis: Start reading a binary value.

Declaration: `procedure ReadBinary(const DestData: TMemoryStream);  Override`

Visibility: public

Description: `ReadBinary` reads a binary valye from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.12 TBinaryObjectReader.ReadFloat

Synopsis: Read a float value

Declaration: `function ReadFloat : Extended;  Override`

Visibility: public

Description: `ReadFloat` reads a float value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.13 TBinaryObjectReader.ReadSingle

Synopsis: Read a single-size float value

Declaration: `function ReadSingle : Single;  Override`

Visibility: public

Description: `ReadSingle` reads a single-sized float value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.14 TBinaryObjectReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency;  Override`

Visibility: public

Description: `var>ReadCurrency` reads a currency-typed value from a binary stream. It is the implementation of the method introduced in TAbstractObjectReader (225).

See also: TAbstractObjectReader.ReadCurrency (229), TBinaryObjectWriter.WriteCurrency (252)

2.33.15 TBinaryObjectReader.ReadDate

Synopsis: Read a date.

Declaration: `function ReadDate : TDateTime;  Override`

Visibility: public

Description: `ReadDate` reads a date value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.16 TBinaryObjectReader.ReadIdent

Synopsis: Read an identifier

Declaration: `function ReadIdent(ValueType: TValueType) : String;  Override`

Visibility: public

Description: `ReadIdent` reads an identifier from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.17 TBinaryObjectReader.ReadInt8

Synopsis: Read an 8-bits integer.

Declaration: `function ReadInt8 : ShortInt;  Override`

Visibility: public

Description: `Read8Int` reads an 8-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.18 TBinaryObjectReader.ReadInt16

Synopsis: Read a 16-bits integer.

Declaration: `function ReadInt16 : SmallInt;  Override`

Visibility: public

Description: `Read16Int` reads a 16-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.19 TBinaryObjectReader.ReadInt32

Synopsis: Read a 32-bits integer.

Declaration: `function ReadInt32 : LongInt;  Override`

Visibility: public

Description: `Read32Int` reads a 32-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.20 TBinaryObjectReader.ReadInt64

Synopsis: Read a 64-bits integer.

Declaration: `function ReadInt64 : Int64;  Override`

Visibility: public

Description: `Read64Int` reads a 64-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.21 TBinaryObjectReader.ReadSet

Synopsis: Read a set

Declaration: `function ReadSet(EnumType: Pointer) : Integer;  Override`

Visibility: public

Description: `ReadSet` reads a set from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.22 TBinaryObjectReader.ReadStr

Synopsis: Read a short string

Declaration: `function ReadStr : String;  Override`

Visibility: public

Description: `ReadStr` reads a short string from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.23 TBinaryObjectReader.ReadString

Synopsis: Read a string

Declaration: `function ReadString(StringType: TValueType) : String;  Override`

Visibility: public

Description: `ReadStr` reads a string of type `StringType` from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.33.24 TBinaryObjectReader.ReadWideString

Synopsis: Read a widestring value from the stream.

Declaration: `function ReadWideString : WideString;  Override`

Visibility: public

Description: `var>ReadWideString` reads a widestring-typed value from a binary stream. It is the implementation of the method introduced in TAbstractObjectReader (225).

See also: TAbstractObjectReader.ReadWideString (232), TBinaryObjectWriter.WriteWideString (253)

2.33.25 TBinaryObjectReader.SkipComponent

Synopsis: Skip a component's data

Declaration: `procedure SkipComponent(SkipComponentInfos: Boolean);  Override`

Visibility: public

Description: `SkipComponent` skips the data of a component in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225).

2.33.26 TBinaryObjectReader.SkipValue

Synopsis: Skip a value's data

Declaration: `procedure SkipValue;  Override`

Visibility: public

Description: `SkipComponent` skips the data of the next value in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (225)

2.34 TBinaryObjectWriter

2.34.1 Description

Driver class which stores component data in binary form.

2.34.2 Method overview

Page	Property	Description
250	BeginCollection	Start writing a collection.
251	BeginComponent	Start writing a component
251	BeginList	Start writing a list.
251	BeginProperty	Start writing a property
250	Create	Creates a new instance of a binary object writer.
250	Destroy	Destroys an instance of the binary object writer.
251	EndList	Mark the end of a list.
251	EndProperty	Marks the end of writing of a property.
251	Write	Write raw data to stream
252	WriteBinary	Writes binary data to the stream.
252	WriteBoolean	Writes a boolean value to the stream.
252	WriteCurrency	Write a currency-valued type to a stream
252	WriteDate	Writes a date type to the stream.
252	WriteFloat	Writes a float value to the stream.
253	WriteIdent	Writes an identifier to the stream.
253	WriteInteger	Writes an integer value to the stream.
253	WriteMethodName	Writes a methodname to the stream.
253	WriteSet	Writes a set value to the stream.
252	WriteSingle	Writes a single-type real value to the stream.
253	WriteString	Writes a string value to the stream.
253	WriteWideString	Write a widestring-valued type to a stream

2.34.3 TBinaryObjectWriter.Create

Synopsis: Creates a new instance of a binary object writer.

Declaration: `constructor Create(Stream: TStream;BufSize: Integer)`

Visibility: public

Description: Creates a new instance of a binary object writer.

2.34.4 TBinaryObjectWriter.Destroy

Synopsis: Destroys an instance of the binary object writer.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys an instance of the binary object writer.

2.34.5 TBinaryObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection;  Override`

Visibility: public

2.34.6 TBinaryObjectWriter.BeginComponent

Synopsis: Start writing a component

Declaration:
```
procedure BeginComponent(Component: TComponent;Flags: TFilerFlags;
                         ChildPos: Integer);  Override
```

Visibility: public

2.34.7 TBinaryObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration:
```
procedure BeginList;  Override
```

Visibility: public

2.34.8 TBinaryObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration:
```
procedure EndList;  Override
```

Visibility: public

2.34.9 TBinaryObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration:
```
procedure BeginProperty(const PropName: String);  Override
```

Visibility: public

2.34.10 TBinaryObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration:
```
procedure EndProperty;  Override
```

Visibility: public

2.34.11 TBinaryObjectWriter.Write

Synopsis: Write raw data to stream

Declaration:
```
procedure Write(const Buffer;Count: LongInt);  Override
```

Visibility: public

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TAbstractObjectWriter.Write (235), TWriter.Write (373)

2.34.12 TBinaryObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer;Count: LongInt);  Override`

Visibility: public

2.34.13 TBinaryObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean);  Override`

Visibility: public

2.34.14 TBinaryObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended);  Override`

Visibility: public

2.34.15 TBinaryObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single);  Override`

Visibility: public

2.34.16 TBinaryObjectWriter.WriteCurrency

Synopsis: Write a currency-valued type to a stream

Declaration: `procedure WriteCurrency(const Value: Currency);  Override`

Visibility: public

Description: `WriteCurrency` writes a currency-typed value to a binary stream. It is the implementation of the method introduced in TAbstractObjectWriter (233).

See also: TAbstractObjectWriter.WriteCurrency (236)

2.34.17 TBinaryObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime);  Override`

Visibility: public

2.34.18 TBinaryObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String);  Override`

Visibility: public

2.34.19 TBinaryObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64);  Override`

Visibility: public

2.34.20 TBinaryObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: String);  Override`

Visibility: public

2.34.21 TBinaryObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt;SetType: Pointer);  Override`

Visibility: public

2.34.22 TBinaryObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: String);  Override`

Visibility: public

2.34.23 TBinaryObjectWriter.WriteWideString

Synopsis: Write a widestring-valued type to a stream

Declaration: `procedure WriteWideString(const Value: WideString);  Override`

Visibility: public

Description: `WriteWidestring` writes a widestring-typed value to a binary stream. It is the implementation of the method introduced in TAbstractObjectWriter (233).

See also: TAbstractObjectWriter.WriteWidestring (237)

2.35 TBits

2.35.1 Description

`TBits` can be used to store collections of bits in an indexed array. This is especially useful for storing collections of booleans: Normally the size of a boolean is the size of the smallest enumerated type, i.e. 1 byte. Since a bit can take 2 values it can be used to store a boolean as well. Since TBits can store 8 bits in a byte, it takes 8 times less space to store an array of booleans in a TBits class then it would take to stoe them in a conventional array.

`TBits` introduces methods to store and retrieve bit values, apply masks, and search for bits.

2.35.2 Method overview

Page	Property	Description
256	AndBits	Performs an `and` operation on the bits.
255	Clear	Clears a particular bit.
256	Clearall	Clears all bits in the array.
254	Create	Creates a new bits collection.
255	Destroy	Destroys a bit collection
258	Equals	Determines whether the bits of 2 arrays are equal.
258	FindFirstBit	Find first bit with a particular value
259	FindNextBit	Searches the next bit with a particular value.
259	FindPrevBit	Searches the previous bit with a particular value.
257	Get	Retrieve the value of a particular bit
255	GetFSize	Returns the number of records used to store the bits.
257	Grow	Expands the bits array to the requested size.
257	NotBits	Performs a `not` operation on the bits.
259	OpenBit	Returns the position of the first bit that is set to `False`.
256	OrBits	Performs an `or` operation on the bits.
258	SetIndex	Sets the start position for FindNextBit (259) and FindPrevBit (259)
255	SetOn	Turn a particular bit on.
256	XorBits	Performs a `xor` operation on the bits.

2.35.3 Property overview

Page	Property	Access	Description
260	Bits	rw	Access to all bits in the array.
260	Size	rw	Current size of the array of bits.

2.35.4 TBits.Create

Synopsis: Creates a new bits collection.

Declaration:
```
constructor Create(TheSize: LongInt);  Virtual
```

Visibility: public

Description: `Create` creates a new bit collection with initial size `TheSize`. The size of the collection can be changed later on.

All bits are initially set to zero.

See also: TBits.Destroy (255)

2.35.5 TBits.Destroy

Synopsis: Destroys a bit collection

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys a previously created bit collection and releases all memory used to store the bit collection.

`Destroy` should never be called directly, `Free` should be used instead.

Errors: None.

See also: TBits.Create (254)

2.35.6 TBits.GetFSize

Synopsis: Returns the number of records used to store the bits.

Declaration: `function GetFSize : LongInt`

Visibility: public

Description: `GetFSize` returns the number of records used to store the current number of bits.

Errors: None.

See also: TBits.Size (260)

2.35.7 TBits.SetOn

Synopsis: Turn a particular bit on.

Declaration: `procedure SetOn(Bit: LongInt)`

Visibility: public

Description: `SetOn` turns on the bit at position `bit`, i.e. sets it to 1. If `bit` is at a position bigger than the current size, the collection is expanded to the required size using Grow (257).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an EBitsError (214) exception is raised.

See also: TBits.Bits (260), TBits.clear (255)

2.35.8 TBits.Clear

Synopsis: Clears a particular bit.

Declaration: `procedure Clear(Bit: LongInt)`

Visibility: public

Description: `Clear` clears the bit at position `bit`. If the array If `bit` is at a position bigger than the current size, the collection is expanded to the required size using Grow (257).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an EBitsError (214) exception is raised.

See also: TBits.Bits (260), TBits.clear (255)

2.35.9 TBits.Clearall

Synopsis: Clears all bits in the array.

Declaration: `procedure Clearall`

Visibility: public

Description: `ClearAll` clears all bits in the array, i.e. sets them to zero. `ClearAll` works faster than clearing all individual bits, since it uses the packed nature of the bits.

Errors: None.

See also: TBits.Bits (260), TBits.clear (255)

2.35.10 TBits.AndBits

Synopsis: Performs an `and` operation on the bits.

Declaration: `procedure AndBits(BitSet: TBits)`

Visibility: public

Description: `andbits` performs an `and` operation on the bits in the array with the bits of array `BitSet`. If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are cleared.

Errors: None.

See also: TBits.clearall (256), TBits.orbits (256), TBits.xorbits (256), TBits.notbits (257)

2.35.11 TBits.OrBits

Synopsis: Performs an `or` operation on the bits.

Declaration: `procedure OrBits(BitSet: TBits)`

Visibility: public

Description: `andbits` performs an `or` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `or` operation is performed.

Errors: None.

See also: TBits.clearall (256), TBits.andbits (256), TBits.xorbits (256), TBits.notbits (257)

2.35.12 TBits.XorBits

Synopsis: Performs a `xor` operation on the bits.

Declaration: `procedure XorBits(BitSet: TBits)`

Visibility: public

Description: `XorBits` performs a `xor` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `xor` operation is performed.

Errors: None.

See also: TBits.clearall (256), TBits.andbits (256), TBits.orbits (256), TBits.notbits (257)

2.35.13 TBits.NotBits

Synopsis: Performs a `not` operation on the bits.

Declaration: `procedure NotBits(BitSet: TBits)`

Visibility: public

Description: `NotBits` performs a `not` operation on the bits in the array with the bits of array `Bitset`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

Errors: None.

See also: TBits.clearall (256), TBits.andbits (256), TBits.orbits (256), TBits.xorbits (256)

2.35.14 TBits.Get

Synopsis: Retrieve the value of a particular bit

Declaration: `function Get(Bit: LongInt) : Boolean`

Visibility: public

Description: `Get` returns `True` if the bit at position `bit` is set, or `False` if it is not set.

Errors: If `bit` is not a valid bit index then an EBitsError (214) exception is raised.

See also: TBits.Bits (260), TBits.FindFirstBit (258), TBits.seton (255)

2.35.15 TBits.Grow

Synopsis: Expands the bits array to the requested size.

Declaration: `procedure Grow(NBit: LongInt)`

Visibility: public

Description: `Grow` expands the bit array so it can at least contain `nbit` bits. If `nbit` is less than the current size, nothing happens.

Errors: If there is not enough memory to complete the operation, then an EBitsError (214)is raised.

See also: TBits.Size (260)

2.35.16 TBits.Equals

Synopsis: Determines whether the bits of 2 arrays are equal.

Declaration: `function Equals(BitSet: TBits) : Boolean`

Visibility: public

Description: `equals` returns `True` if all the bits in `BitSet` are the same as the ones in the current BitSet; if not, `False` is returned.

If the sizes of the two BitSets are different, the arrays are still reported equal when all the bits in the larger set, which are not present in the smaller set, are zero.

Errors: None.

See also: TBits.clearall (256), TBits.andbits (256), TBits.orbits (256), TBits.xorbits (256)

2.35.17 TBits.SetIndex

Synopsis: Sets the start position for FindNextBit (259) and FindPrevBit (259)

Declaration: `procedure SetIndex(Index: LongInt)`

Visibility: public

Description: `SetIndex` sets the search start position forFindNextBit (259) and FindPrevBit (259) to `Index`. This means that these calls will start searching from position `Index`.

This mechanism provides an alternative to FindFirstBit (258) which can also be used to position for the `FindNextBit` and `FindPrevBit` calls.

Errors: None.

See also: TBits.FindNextBit (259), TBits.FindPrevBit (259), TBits.FindFirstBit (258), TBits.OpenBit (259)

2.35.18 TBits.FindFirstBit

Synopsis: Find first bit with a particular value

Declaration: `function FindFirstBit(State: Boolean) : LongInt`

Visibility: public

Description: `FindFirstBit` searches for the first bit with value `State`. It returns the position of this bit, or `-1` if no such bit was found.

The search starts at position 0 in the array. If the first search returned a positive result, the found position is saved, and the FindNextBit (259) and FindPrevBit (259) will use this position to resume the search. To start a search from a certain position, the start position can be set with the SetIndex (258) instead.

Errors: None.

See also: TBits.FindNextBit (259), TBits.FindPrevBit (259), TBits.OpenBit (259), TBits.SetIndex (258)

2.35.19 TBits.FindNextBit

Synopsis: Searches the next bit with a particular value.

Declaration: `function FindNextBit : LongInt`

Visibility: public

Description: `FindNextBit` resumes a previously started search. It searches for the next bit with the value specified in the FindFirstBit (258). The search is done towards the end of the array and starts at the position last reported by one of the `Find` calls or at the position set with SetIndex (258).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: TBits.FindFirstBit (258), TBits.FindPrevBit (259), TBits.OpenBit (259), TBits.SetIndex (258)

2.35.20 TBits.FindPrevBit

Synopsis: Searches the previous bit with a particular value.

Declaration: `function FindPrevBit : LongInt`

Visibility: public

Description: `FindPrevBit` resumes a previously started search. It searches for the previous bit with the value specified in the FindFirstBit (258). The search is done towards the beginning of the array and starts at the position last reported by one of the `Find` calls or at the position set with SetIndex (258).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: TBits.FindFirstBit (258), TBits.FindNextBit (259), TBits.OpenBit (259), TBits.SetIndex (258)

2.35.21 TBits.OpenBit

Synopsis: Returns the position of the first bit that is set to `False`.

Declaration: `function OpenBit : LongInt`

Visibility: public

Description: `OpenBit` returns the position of the first bit whose value is `0` (`False`), or `-1` if no open bit was found. This call is equivalent to `FindFirstBit(False)`, except that it doesn't set the position for the next searches.

Errors: None.

See also: TBits.FindFirstBit (258), TBits.FindPrevBit (259), TBits.FindFirstBit (258), TBits.SetIndex (258)

2.35.22 TBits.Bits

Synopsis: Access to all bits in the array.

Declaration: `Property Bits[Bit: LongInt]: Boolean; default`

Visibility: public

Access: Read,Write

Description: `Bits` allows indexed access to all of the bits in the array. It gives `True` if the bit is `1`, `False` otherwise; Assigning to this property will set, respectively clear the bit.

Errors: If an index is specified which is out of the allowed range then an EBitsError (214) exception is raised.

See also: TBits.Size (260)

2.35.23 TBits.Size

Synopsis: Current size of the array of bits.

Declaration: `Property Size : LongInt`

Visibility: public

Access: Read,Write

Description: `Size` is the current size of the bit array. Setting this property will adjust the size; this is equivalent to calling `Grow(Value-1)`

Errors: If an invalid size (negative or too large) is specified, a EBitsError (214) exception is raised.

See also: TBits.Bits (260)

2.36 TCollection

2.36.1 Description

`TCollection` implements functionality to manage a collection of named objects. Each of these objects needs to be a descendent of the TCollectionItem (265) class. Exactly which type of object is managed can be seen from the TCollection.ItemClass (264) property.

Normally, no `TCollection` is created directly. Instead, a descendents of `TCollection` and TCollectionItem (265) are created as a pair.

2.36.2 Method overview

Page	Property	Description
262	Add	Creates and adds a new item to the collection.
262	Assign	Assigns one collection to another.
262	BeginUpdate	Start an update batch.
263	Clear	Removes all items from the collection.
261	Create	Creates a new collection.
263	Delete	Delete an item from the collection.
261	Destroy	Destroys the collection and frees all the objects it manages.
263	EndUpdate	Ends an update batch.
264	FindItemID	Searches for an Item in the collection, based on its TCollectionItem.ID (266) property.
263	Insert	Insert an item in the collection.
261	Owner	Owner of the collection.

2.36.3 Property overview

Page	Property	Access	Description
264	Count	r	Number of items in the collection.
264	ItemClass	r	Class pointer for each item in the collection.
265	Items	rw	Indexed array of items in the collection.

2.36.4 TCollection.Create

Synopsis: Creates a new collection.

Declaration: `constructor Create(AItemClass: TCollectionItemClass)`

Visibility: public

Description: `Create` instantiates a new instance of the `TCollection` class which will manage objects of class `AItemClass`. It creates the list used to hold all objects, and stores the `AItemClass` for the adding of new objects to the collection.

See also: TCollection.ItemClass (264), TCollection.Destroy (261)

2.36.5 TCollection.Destroy

Synopsis: Destroys the collection and frees all the objects it manages.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` first clears the collection, and then frees all memory allocated to this instance.

Don't call `Destroy` directly, call `Free` instead.

See also: TCollection.Create (261)

2.36.6 TCollection.Owner

Synopsis: Owner of the collection.

Declaration: `function Owner : TPersistent`

Visibility: public

Description: `Owner` returns a reference to the owner of the collection. This property is required by the object inspector to be able to show the collection.

2.36.7 TCollection.Add

Synopsis: Creates and adds a new item to the collection.

Declaration: `function Add : TCollectionItem`

Visibility: public

Description: `Add` instantiates a new item of class TCollection.ItemClass (264) and adds it to the list. The newly created object is returned.

See also: TCollection.ItemClass (264), TCollection.Clear (263)

2.36.8 TCollection.Assign

Synopsis: Assigns one collection to another.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` assigns the contents of one collection to another. It does this by clearing the items list, and adding as much elements as there are in the `Source` collection; it assigns to each created element the contents of it's counterpart in the `Source` element.

Two collections cannot be assigned to each other if instances of the `ItemClass` classes cannot be assigned to each other.

Errors: If the objects in the collections cannot be assigned to one another, then an `EConvertError` is raised.

See also: TPersistent.Assign (317), TCollectionItem (265)

2.36.9 TCollection.BeginUpdate

Synopsis: Start an update batch.

Declaration: `procedure BeginUpdate;  Virtual`

Visibility: public

Description: `BeginUpdate` is called at the beginning of a batch update. It raises the update count with 1.

Call `BeginUpdate` at the beginning of a series of operations that will change the state of the collection. This will avoid the call to TCollection.Update (260) for each operation. At the end of the operations, a corresponding call to `EndUpdate` must be made. It is best to do this in the context of a `Try ... finally` block:

```
With MyCollection Do
    try
      BeginUpdate;
      // Some Lengthy operations
    finally
      EndUpdate;
    end;
```

This insures that the number of calls to `BeginUpdate` always matches the number of calls to TCollection.EndUpdate (263), even in case of an exception.

See also: TCollection.EndUpdate (263), TCollection.Changed (260), TCollection.Update (260)

2.36.10 TCollection.Clear

Synopsis: Removes all items from the collection.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` will clear the collection, i.e. each item in the collection is destroyed and removed from memory. After a call to `Clear`, `Count` is zero.

See also: TCollection.Add (262), TCollectionItem.Destroy (266), TCollection.Destroy (261)

2.36.11 TCollection.EndUpdate

Synopsis: Ends an update batch.

Declaration: `procedure EndUpdate;  Virtual`

Visibility: public

Description: `EndUpdate` signals the end of a series of operations that change the state of the collection, possibly triggering an update event. It does this by decreasing the update count with 1 and calling TCollection.Changed (260) it should always be used in conjunction with TCollection.BeginUpdate (262), preferably in the `Finally` section of a `Try ... Finally` block.

See also: TCollection.BeginUpdate (262), TCollection.Changed (260), TCollection.Update (260)

2.36.12 TCollection.Delete

Synopsis: Delete an item from the collection.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the item at (zero based) position `Index` from the collection. This will result in a `cnDeleted` notification.

Errors: If an invalid index is specified, an exception is raised.

See also: TCollection.Items (265), TCollection.Insert (263), TCollection.Clear (263)

2.36.13 TCollection.Insert

Synopsis: Insert an item in the collection.

Declaration: `function Insert(Index: Integer) : TCollectionItem`

Visibility: public

Description: `Insert` creates a new item instance and inserts it in the collection at position `Index`, and returns the new instance.

In contrast, TCollection.Add (262) adds a new item at the end.

Errors: None.

See also: TCollection.Add (262), TCollection.Delete (263), TCollection.Items (265)

2.36.14 TCollection.FindItemID

Synopsis: Searches for an Item in the collection, based on its TCollectionItem.ID (266) property.

Declaration: `function FindItemID(ID: Integer) : TCollectionItem`

Visibility: public

Description: `FindItemID` searches through the collection for the item that has a value of `ID` for its TCollectionItem.ID (266) property, and returns the found item. If no such item is found in the collection, `Nil` is returned.

The routine performs a linear search, so this can be slow on very large collections.

See also: TCollection.Items (265), TCollectionItem.ID (266)

2.36.15 TCollection.Count

Synopsis: Number of items in the collection.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` contains the number of items in the collection.

Remark: The items in the collection are identified by their TCollectionItem.Index (267) property, which is a zero-based index, meaning that it can take values between `0` and `Count`.

See also: TCollectionItem.Index (267), TCollection.Items (265)

2.36.16 TCollection.ItemClass

Synopsis: Class pointer for each item in the collection.

Declaration: `Property ItemClass : TCollectionItemClass`

Visibility: public

Access: Read

Description: `ItemClass` is the class pointer with which each new item in the collection is created. It is the value that was passed to the collection's constructor when it was created, and does not change during the lifetime of the collection.

See also: TCollectionItem (265), TCollection.Items (265)

2.36.17 TCollection.Items

Synopsis: Indexed array of items in the collection.

Declaration: `Property Items[Index: Integer]: TCollectionItem`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the collection. Since the array is zero-based, `Index` should be an integer between `0` and `Count-1`.

It is possible to set or retrieve an element in the array. When setting an element of the array, the object that is assigned should be compatible with the class of the objects in the collection, as given by the TCollection.ItemClass (264) property.

Adding an element to the array can be done with the TCollection.Add (262) method. The array can be cleared with the TCollection.Clear (263) method. Removing an element of the array should be done by freeing that element.

See also: TCollection.Count (264), TCollection.ItemClass (264), TCollection.Clear (263), TCollection.Add (262)

2.37 TCollectionItem

2.37.1 Description

`TCollectionItem` and TCollection (260) form a pair of base classes that manage a collection of named objects. The `TCollectionItem` is the named object that is managed, it represents one item in the collection. An item in the collection is represented by three properties: TCollectionItem.DisplayName (267), TCollection.Index (260) and TCollectionItem.ID (266).

A `TCollectionItem` object is never created directly. To manage a set of named items, it is necessary to make a descendent of `TCollectionItem` to which needed properties and methods are added. This descendant can then be managed with a TCollection (260) class. The managing collection will create and destroy it's items by itself, it should therefore never be necessary to create `TCollectionItem` descendents manually.

2.37.2 Method overview

Page	Property	Description
266	Create	Creates a new instance of this collection item.
266	Destroy	Destroyes this collection item.

2.37.3 Property overview

Page	Property	Access	Description
266	Collection	rw	Pointer to the collection managing this item.
267	DisplayName	rw	Name of the item, displayed in the object inspector.
266	ID	r	Initial index of this item.
267	Index	rw	Index of the item in its managing collection TCollection.Items (265) property.

2.37.4 TCollectionItem.Create

Synopsis: Creates a new instance of this collection item.

Declaration: `constructor Create(ACollection: TCollection); Virtual`

Visibility: public

Description: `Create` instantiates a new item in a TCollection (260). It is called by the TCollection.Add (262) function and should under normal circumstances never be called directly. called

See also: TCollectionItem.Destroy (266)

2.37.5 TCollectionItem.Destroy

Synopsis: Destroyes this collection item.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` removes the item from the managing collection and Destroys the item instance.

This is the only way to remove items from a collection;

See also: TCollectionItem.Create (266)

2.37.6 TCollectionItem.Collection

Synopsis: Pointer to the collection managing this item.

Declaration: `Property Collection : TCollection`

Visibility: public

Access: Read,Write

Description: `Collection` points to the collection managing this item. This property can be set to point to a new collection. If this is done, the old collection will be notified that the item should no longer be managed, and the new collection is notified that it should manage this item as well.

See also: TCollection (260)

2.37.7 TCollectionItem.ID

Synopsis: Initial index of this item.

Declaration: `Property ID : Integer`

Visibility: public

Access: Read

Description: `ID` is the initial value of TCollectionItem.Index (267); it doesn't change after the index changes. It can be used to uniquely identify the item. The `ID` property doesn't change as items are added and removed from the collection.

While the TCollectionItem.Index (267) property forms a continuous series, `ID` does not. If items are removed from the collection, their `ID` is not used again, leaving gaps. Only when the collection is initialiy created, the `ID` and `Index` properties will be equal.

See also: TCollection.Items (265), TCollectionItem.Index (267)

2.37.8 TCollectionItem.Index

Synopsis: Index of the item in its managing collection TCollection.Items (265) property.

Declaration: `Property Index : Integer`

Visibility: public

Access: Read,Write

Description: `Index` is the current index of the item in its managing collection's TCollection.Items (265) property. This property may change as items are added and removed from the collection.

The index of an item is zero-based, i.e. the first item has index zero. The last item has index `Count-1` where `Count` is the number of items in the collection.

The `Index` property of the items in a collection form a continuous series ranging from `0` to `Count-1`. The TCollectionItem.ID (266) property does not form a continuous series, but can also be used to identify an item.

See also: TCollectionItem.ID (266), TCollection.Items (265)

2.37.9 TCollectionItem.DisplayName

Synopsis: Name of the item, displayed in the object inspector.

Declaration: `Property DisplayName : String`

Visibility: public

Access: Read,Write

Description: `DisplayName` contains the name of this item as shown in the object inspector. For `TCollectionItem` this returns always the class name of the managing collection, followed by the index of the item.

`TCollectionItem` does not implement any functionality to store the `DisplayName` property. The property can be set, but this will have no effect other than that the managing collection is notified of a change. The actual displayname will remain unchanged. To store the `DisplayName` property,`TCollectionItem` descendants should override the TCollectionItem.SetDisplayName (265) and TCollectionItem.GetDisplayName (265) to add storage functionality.

See also: TCollectionItem.Index (267), TCollectionItem.ID (266), TCollectionItem.GetDisplayName (265), TCollectionItem.SetDisplayName (265)

2.38 TComponent

2.38.1 Description

`TComponent` is the base class for any set of classes that needs owner-owned functionality, and which needs support for property streaming. All classes that should be handled by an IDE (Integrated Development Environment) must descend from `TComponent`, as it includes all support for streaming all its published properties.

Components can 'own' other components. `TComponent` introduces methods for enumerating the child components. It also allows to name the owned components with a unique name. Furthermore, functionality for sending notifications when a component is removed from the list or removed from memory alltogether is also introduced in `TComponent`

`TComponent` introduces a form of automatic memory management: When a component is destroyed, all its child components will be destroyed first.

2.38.2 Method overview

Page	Property	Description
269	BeforeDestruction	Overrides standard BeforeDestruction.
269	Create	Creates a new instance of the component.
269	Destroy	Destroys the instance of the component.
269	DestroyComponents	Destroy child components.
270	Destroying	Called when the component is being destroyed
270	ExecuteAction	Standard action execution method.
270	FindComponent	Finds and returns the named component in the owned components.
270	FreeNotification	Ask the component to notify called when it is being destroyed.
271	FreeOnRelease	Part of the `IVCLComObject` interface.
271	GetParentComponent	Returns the parent component.
271	HasParent	Does the component have a parent ?
272	InsertComponent	Insert the given component in the list of owned components.
273	IsImplementorOf	Checks if the current component is the implementor of the interface
273	ReferenceInterface	Interface implementation of Notification
272	RemoveComponent	Remove the given component from the list of owned components.
271	RemoveFreeNotification	Remove a component from the Free Notification list.
272	SafeCallException	Part of the `IVCLComObject` Interface.
272	SetSubComponent	Sets the `csSubComponent` style.
273	UpdateAction	Updates the state of an action.
268	WriteState	Writes the component to a stream.

2.38.3 Property overview

Page	Property	Access	Description
274	ComponentCount	r	Count of owned components
274	ComponentIndex	rw	Index of component in it's owner's list.
273	Components	r	Indexed list (zero-based) of all owned components.
274	ComponentState	r	Current component's state.
275	ComponentStyle	r	Current component's style.
275	DesignInfo	rw	Information for IDE designer.
276	Name	rws	Name of the component.
275	Owner	r	Owner of this component.
276	Tag	rw	Tag value of the component.
275	VCLComObject	rw	Not implemented.

2.38.4 TComponent.WriteState

Synopsis: Writes the component to a stream.

Declaration: `procedure WriteState(Writer: TWriter);  Virtual`

Visibility: public

Description: `WriteState` writes the component's current state to a stream through the writer (372) object `writer`. Values for all published properties of the component can be written to the stream. Normally there is no need to call `WriteState` directly. The streaming system calls `WriteState` itself.

The TComponent (267) implementation of `WriteState` simply calls TWriter.WriteData (372). Descendent classes can, however, override `WriteState`to provide additional processing of stream data.

See also: TComponent.ReadState (267), TStream.WriteComponent (336), TWriter.WriteData (372)

2.38.5 TComponent.Create

Synopsis: Creates a new instance of the component.

Declaration: `constructor Create(AOwner: TComponent); Virtual`

Visibility: public

Description: `Create` creates a new instance of a `TComponent` class. If `AOwner` is not `Nil`, the new component attempts to insert itself in the list of owned components of the owner.

See also: TComponent.Insert (267), TComponent.Owner (275)

2.38.6 TComponent.BeforeDestruction

Synopsis: Overrides standard BeforeDestruction.

Declaration: `procedure BeforeDestruction; Override`

Visibility: public

Description: `BeforeDestruction` is overridden by `TComponent` to set the `csDestroying` flag in ComponentState (185)

See also: ComponentState (185)

2.38.7 TComponent.Destroy

Synopsis: Destroys the instance of the component.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` sends a `opRemove` notification to all components in the free-notification list. After that, all owned components are destroyed by calling DestroyComponents (269) (and hence removed from the list of owned components). When this is done, the component removes itself from its owner's child component list. After that, the parent's destroy method is called.

See also: TComponent.Notification (267), TComponent.Owner (275), TComponent.DestroyComponents (269), TComponent.Components (273)

2.38.8 TComponent.DestroyComponents

Synopsis: Destroy child components.

Declaration: `procedure DestroyComponents`

Visibility: public

Description: `DestroyComponents` calls the destructor of all owned components, till no more components are left in the Components (273) array.

Calling the destructor of an owned component has as the effect that the component will remove itself from the list of owned components, if nothing has disrupted the sequence of destructors.

Errors: If an overridden 'destroy' method does not call it's intherited destructor or raises an exception, it's TComponent.Destroy (269) destructor will not be called, which may result in an endless loop.

See also: TComponent.Destroy (269), TComponent.Components (273)

2.38.9 TComponent.Destroying

Synopsis: Called when the component is being destroyed

Declaration: `procedure Destroying`

Visibility: public

Description: `Destroying` sets the `csDestroying` flag in the component's state (267) property, and does the same for all owned components.

It is not necessary to call `Destroying` directly, the destructor Destroy (269) does this automatically.

See also: TComponent.State (267), TComponent.Destroy (269)

2.38.10 TComponent.ExecuteAction

Synopsis: Standard action execution method.

Declaration: `function ExecuteAction(Action: TBasicAction) : Boolean;  Dynamic`

Visibility: public

Description: `ExecuteAction` checks whether `Action` handles the current component, and if yes, calls the `ExecuteAction` method, passing itself as a parameter. The function returns `True` if the action handles the current component.

See also: TBasicAction (237), TBasicAction.ExecuteAction (237), TBasicAction.HandlesTarget (238), UpdateAction (185)

2.38.11 TComponent.FindComponent

Synopsis: Finds and returns the named component in the owned components.

Declaration: `function FindComponent(const AName: String) : TComponent`

Visibility: public

Description: `FindComponent` searches the component with name `AName` in the list of owned components. If `AName` is empty, then `Nil` is returned.

See also: TComponent.Components (273), TComponent.Name (276)

2.38.12 TComponent.FreeNotification

Synopsis: Ask the component to notify called when it is being destroyed.

Declaration: `procedure FreeNotification(AComponent: TComponent)`

Visibility: public

Description: `FreeNotification` inserts `AComponent` in the freenotification list. When the component is destroyed, the Notification (267) method is called for all components in the freenotification list.

See also: TComponent.Components (273), TComponent.Notification (267)

2.38.13 TComponent.RemoveFreeNotification

Synopsis: Remove a component from the Free Notification list.

Declaration: `procedure RemoveFreeNotification(AComponent: TComponent)`

Visibility: public

Description: `RemoveFreeNotification` removes `AComponent` from the freenotification list.

See also: FreeNotification (185)

2.38.14 TComponent.FreeOnRelease

Synopsis: Part of the `IVCLComObject` interface.

Declaration: `procedure FreeOnRelease`

Visibility: public

Description: Provided for Delphi compatibility, but is not yet impltmentedd.

2.38.15 TComponent.GetParentComponent

Synopsis: Returns the parent component.

Declaration: `function GetParentComponent : TComponent;  Dynamic`

Visibility: public

Description: `GetParentComponent` can be implemented to return the parent component of this component. The implementation of this method in `TComponent` always returns `Nil`. Descendent classes must override this method to return the visual parent of the component.

See also: TComponent.HasParent (271), TComponent.Owner (275)

2.38.16 TComponent.HasParent

Synopsis: Does the component have a parent ?

Declaration: `function HasParent : Boolean;  Dynamic`

Visibility: public

Description: `HasParent` can be implemented to return whether the parent of the component exists. The implementation of this method in TComponent always returns `False`, and should be overridden by descendent classes to return `True` when a parent is available. If `HasParent` returns `True`, then GetParentComponent (271) will return the parent component.

See also: TComponent.HasParent (271), TComponent.Owner (275)

2.38.17 TComponent.InsertComponent

Synopsis: Insert the given component in the list of owned components.

Declaration:
```
procedure InsertComponent(AComponent: TComponent)
```

Visibility: public

Description: `InsertComponent` attempts to insert `AComponent` in the list with owned components. It first calls ValidateComponent (267) to see whether the component can be inserted. It then checks whether there are no name conflicts by calling ValidateRename (267). If neither of these checks have raised an exception the component is inserted, and notified of the insert.

See also: TComponent.RemoveComponent (272), TComponent.Insert (267), TComponent.ValidateContainer (267), TComponent.ValidateRename (267), TComponent.Notification (267)

2.38.18 TComponent.RemoveComponent

Synopsis: Remove the given component from the list of owned components.

Declaration:
```
procedure RemoveComponent(AComponent: TComponent)
```

Visibility: public

Description: `RemoveComponent` will send an `opRemove` notification to `AComponent` and will then proceed to remove `AComponent` from the list of owned components.

See also: TComponent.InsertComponent (272), TComponent.Remove (267), TComponent.ValidateRename (267), TComponent.Notification (267)

2.38.19 TComponent.SafeCallException

Synopsis: Part of the `IVCLComObject` Interface.

Declaration:
```
function SafeCallException(ExceptObject: TObject;ExceptAddr: Pointer)
                                  : Integer;  Override
```

Visibility: public

Description: Provided for Delphi compatibility, but not implemented.

2.38.20 TComponent.SetSubComponent

Synopsis: Sets the `csSubComponent` style.

Declaration:
```
procedure SetSubComponent(ASubComponent: Boolean)
```

Visibility: public

Description: `SetSubComponent` includes `csSubComponent` in the ComponentStyle (275) property if `ASubComponent` is `True`, and excludes it again if `ASubComponent` is `False`.

See also: TComponent.ComponentStyle (275)

2.38.21 TComponent.UpdateAction

Synopsis: Updates the state of an action.

Declaration: `function UpdateAction(Action: TBasicAction) : Boolean; Dynamic`

Visibility: public

Description: `UpdateAction` checks whether `Action` handles the current component, and if yes, calls the `UpdateTarget` method, passing itself as a parameter. The function returns `True` if the action handles the current component.

See also: TBasicAction (237), TBasicAction.UpdateTarget (239), TBasicAction.HandlesTarget (238), ExecuteAction (185)

2.38.22 TComponent.IsImplementorOf

Synopsis: Checks if the current component is the implementor of the interface

Declaration: `function IsImplementorOf(const Intf: IInterface) : Boolean`

Visibility: public

Description: `IsImplementorOf` returns `True` if the current component implements the given interface. The interface should descend from IInterfaceComponentReference (218) and the `GetComponent` method should return the current instance.

See also: IInterfaceComponentReference (218)

2.38.23 TComponent.ReferenceInterface

Synopsis: Interface implementation of Notification

Declaration: `procedure ReferenceInterface(const intf: IInterface;op: TOperation)`

Visibility: public

Description: `ReferenceInterface` can be used to notify an interface of a component operation: it is the equivalent of the TComponent.Notification (267) method of `TComponent` for interfaces. If the interface implements IInterfaceComponentReference (218), then the component that implements it is notified of the given operation `Op`.

Errors: None.

See also: TComponent.Notification (267), IInterfaceComponentReference (218)

2.38.24 TComponent.Components

Synopsis: Indexed list (zero-based) of all owned components.

Declaration: `Property Components[Index: Integer]: TComponent`

Visibility: public

Access: Read

Description: `Components` provides indexed access to the list of owned components. `Index` can range from 0 to ComponentCount-1 (274).

See also: TComponent.ComponentCount (274), TComponent.Owner (275)

2.38.25 TComponent.ComponentCount

Synopsis: Count of owned components

Declaration: `Property ComponentCount : Integer`

Visibility: public

Access: Read

Description: `ComponentCount` returns the number of components that the current component owns. It can be used to determine the valid index range in the Component (273) array.

See also: TComponent.Components (273), TComponent.Owner (275)

2.38.26 TComponent.ComponentIndex

Synopsis: Index of component in it's owner's list.

Declaration: `Property ComponentIndex : Integer`

Visibility: public

Access: Read,Write

Description: `ComponentIndex` is the index of the current component in its owner's list of components. If the component has no owner, the value of this property is -1.

See also: TComponent.Components (273), TComponent.ComponentCount (274), TComponent.Owner (275)

2.38.27 TComponent.ComponentState

Synopsis: Current component's state.

Declaration: `Property ComponentState : TComponentState`

Visibility: public

Access: Read

Description: `ComponentState` indicates the current state of the component. It is a set of flags which indicate the various stages in the lifetime of a component. The following values can occur in this set:

Table 2.17: Component states

Flag	Meaning
csLoading	The component is being loaded from stream
csReading	Component properties are being read from stream.
csWriting	Component properties are weing written to stream.
csDestroying	The component or one of it's owners is being destoyed.
csAncestor	The component is being streamed as part of a frame
csUpdating	The component is being updated
csFixups	References to other components are being resolved
csFreeNotification	The component has freenotifications.
csInline	The component is being loaded as part of a frame
csDesignInstance	? not used.

The component state is set by various actions such as reading it from stream, destroying it etc.

See also: TComponent.SetAncestor (267), TComponent.SetDesigning (267), TComponent.SetInline (267), TComponent.SetDesignInstance (267), TComponent.Updating (267), TComponent.Updated (267), TComponent.Loaded (267)

2.38.28 TComponent.ComponentStyle

Synopsis: Current component's style.

Declaration: `Property ComponentStyle : TComponentStyle`

Visibility: public

Access: Read

Description: Current component's style.

2.38.29 TComponent.DesignInfo

Synopsis: Information for IDE designer.

Declaration: `Property DesignInfo : LongInt`

Visibility: public

Access: Read,Write

Description: `DesignInformation` can be used by an IDE to store design information in the component. It should not be used by an application programmer.

See also: TComponent.Tag (276)

2.38.30 TComponent.Owner

Synopsis: Owner of this component.

Declaration: `Property Owner : TComponent`

Visibility: public

Access: Read

Description: `Owner` returns the owner of this component. The owner cannot be set except by explicitly inserting the component in another component's owned components list using that component's InsertComponent (272) method, or by removing the component from it's owner's owned component list using the RemoveComponent (272) method.

See also: TComponent.Components (273), TComponent.InsertComponent (272), TComponent.RemoveComponent (272)

2.38.31 TComponent.VCLComObject

Synopsis: Not implemented.

Declaration: `Property VCLComObject : Pointer`

Visibility: public

Access: Read,Write

Description: `VCLComObject` is not yet implemented in Free Pascal.

2.38.32 TComponent.Name

Synopsis: Name of the component.

Declaration: `Property Name : TComponentName`

Visibility: published

Access: Read,Write

Description: `Name` is the name of the component. This name should be a valid identifier, i.e. must start with a letter, and can contain only letters, numbers and the underscore character. When attempting to set the name of a component, the name will be checked for validity. Furthermore, when a component is owned by another component, the name must be either empty or must be unique among the child component names.

Errors: Attempting to set the name to an invalid value will result in an exception being raised.

See also: TComponent.ValidateRename (267), TComponent.Owner (275)

2.38.33 TComponent.Tag

Synopsis: Tag value of the component.

Declaration: `Property Tag : LongInt`

Visibility: published

Access: Read,Write

Description: `Tag` can be used to store an integer value in the component. This value is streamed together with all other published properties. It can be used for instance to quickly identify a component in an event handler.

See also: TComponent.Name (276)

2.39 TCustomMemoryStream

2.39.1 Description

`TCustomMemoryStream` is the parent class for streams that stored their data in memory. It introduces all needed functions to handle reading from and navigating through the memory, and introduces a Memory (278) property which points to the memory area where the stream data is kept.

The only thing which `TCustomMemoryStream` does not do is obtain memory to store data when writing data or the writing of data. This functionality is implemented in descendent streams such as TMemoryStream (306). The reason for this approach is that this way it is possible to create e.g. read-only descendents of `TCustomMemoryStream` that point to a fixed part in memory which can be read from, but not written to.

Remark: Since `TCustomMemoryStream` is an abstract class, do not create instances of `TMemoryStream` directly. Instead, create instances of descendents such as TMemoryStream (306).

2.39.2 Method overview

Page	Property	Description
277	GetSize	return the size of the stream.
277	Read	Reads `Count` bytes from the stream into `buffer`.
278	SaveToFile	Writes the contents of the stream to a file.
278	SaveToStream	Writes the contents of the memory stream to another stream.
277	Seek	Sets a new position in the stream.

2.39.3 Property overview

Page	Property	Access	Description
278	Memory	r	Pointer to the data kept in the memory stream.

2.39.4 TCustomMemoryStream.GetSize

Synopsis: return the size of the stream.

Declaration: `function GetSize : Int64;  Override`

Visibility: public

Description: `GetSize` returns the size of the reseved memory. It should not be used directly.

See also: TStream.Size (340)

2.39.5 TCustomMemoryStream.Read

Synopsis: Reads `Count` bytes from the stream into `buffer`.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` reads `Count` bytes from the stream into the memory pointed to by `buffer`. It returns the number of bytes actually read.

This method overrides the TStream.Read (333) method of TStream (332). It will read as much bytes as are still available in the memory area pointer to by Memory (278). After the bytes are read, the internal stream position is updated.

See also: TCustomMemoryStream.Memory (278), TStream.Read (333)

2.39.6 TCustomMemoryStream.Seek

Synopsis: Sets a new position in the stream.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` overrides the abstract TStream.Seek (333) method. It simply updates the internal stream position, and returns the new position.

Errors: No checking is done whether the new position is still a valid position, i.e. whether the position is still within the range `0..Size`. Attempting a seek outside the valid memory range of the stream may result in an exception at the next read or write operation.

See also: TStream.Position (340), TStream.Size (340), TCustomMemoryStream.Memory (278)

2.39.7 TCustomMemoryStream.SaveToStream

Synopsis: Writes the contents of the memory stream to another stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: public

Description: `SaveToStream` writes the contents of the memory stream to `Stream`. The content of `Stream` is not cleared first. The current position of the memory stream is not changed by this action.

Remark: This method will work much faster than the use of the TStream.CopyFrom (335) method:

```
Seek(0,soFromBeginning);
  Stream.CopyFrom(Self,Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToStream` writes the contents of the memory as one big block.

Errors: If an error occurs when writing to `Stream` an EStreamError (216) exception will be raised.

See also: TCustomMemoryStream.SaveToFile (278), TStream.CopyFrom (335)

2.39.8 TCustomMemoryStream.SaveToFile

Synopsis: Writes the contents of the stream to a file.

Declaration: `procedure SaveToFile(const FileName: String)`

Visibility: public

Description: `SaveToFile` writes the contents of the stream to a file with name `FileName`. It simply creates a filestream and writes the contents of the memorystream to this file stream using TCustomMemoryStream.SaveToStream (278).

Remark: This method will work much faster than the use of the TStream.CopyFrom (335) method:

```
Stream:=TFileStream.Create(fmCreate,FileName);
  Seek(0,soFromBeginning);
  Stream.CopyFrom(Self,Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToFile` writes the contents of the memory as one big block.

Errors: If an error occurs when creating or writing to the file, an EStreamError (216) exception may occur.

See also: TCustomMemoryStream.SaveToStream (278), TFileStream (284), TStream.CopyFrom (335)

2.39.9 TCustomMemoryStream.Memory

Synopsis: Pointer to the data kept in the memory stream.

Declaration: `Property Memory : Pointer`

Visibility: public

Access: Read

Description: `Memory` points to the memory area where stream keeps it's data. The property is read-only, so the pointer cannot be set this way.

Remark: Do not write to the memory pointed to by `Memory`, since the memory content may be read-only, and thus writing to it may cause errors.

See also: TStream.Size (340)

2.40 TDataModule

2.40.1 Description

`TDataModule` is a container for non-visual objects which can be used in an IDE to group non-visual objects which can be used by various other containers (forms) in a project. Notably, data access components are typically stored on a datamodule. Web components and services can also be implemented as descendents of datamodules.

`TDataModule` introduces some events which make it easier to program, and provides the needed streaming capabilities for persistent storage.

An IDE will typically allow to create a descendent of `TDataModule` which contains non-visual components in it's published property list.

2.40.2 Method overview

Page	Property	Description
280	AfterConstruction	Overrides standard TObject (185) behaviour.
280	BeforeDestruction	
279	Create	Create a new instance of a `TDataModule`.
280	CreateNew	
280	Destroy	Destroys the `TDataModule` instance.

2.40.3 Property overview

Page	Property	Access	Description
281	DesignOffset	rw	Position property needed for manipulation in an IDE.
281	DesignSize	rw	Size property needed for manipulation in an IDE.
282	OldCreateOrder	rw	Determines when `OnCreate` and `OnDestroy` are triggered.
281	OnCreate	rw	Event handler, called when the datamodule is created.
282	OnDestroy	rw	Event handler, called when the datamodule is destroyed.

2.40.4 TDataModule.Create

Synopsis: Create a new instance of a `TDataModule`.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` creates a new instance of the `TDatamodule` and calls TDatamodule.CreateNew (280). After that it reads the published properties from a stream using InitInheritedComponent (205) if a descendent class is instantiated. If the OldCreateOrder (282) property is `True`, the OnCreate (185) event is called.

Errors: An exception can be raised during the streaming operation.

See also: TDataModule.CreateNew (280)

2.40.5 TDataModule.CreateNew

Synopsis:

Declaration:
```
constructor CreateNew(AOwner: TComponent)
constructor CreateNew(AOwner: TComponent;CreateMode: Integer);  Virtual
```

Visibility: public

Description: `CreateNew` creates a new instance of the class, but bypasses the streaming mechanism. The `CreateMode` parameter (by default zero) is not used in `TDataModule`. If the AddDataModule (199) handler is set, then it is called, with the newly created instance as an argument.

See also: TDataModule.Create (279), AddDataModule (199), TDataModule.OnCreate (281)

2.40.6 TDataModule.Destroy

Synopsis: Destroys the `TDataModule` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` destroys the `TDataModule` instance. If the OldCreateOrder (282) property is `True` the OnDestroy (282) event handler is called prior to destroying the data module.

Before calling the inherited destroy, the RemoveDataModule (199) handler is called if it is set, and `Self` is passed as a parameter.

Errors: An event can be raised during the `OnDestroy` event handler.

See also: TDataModule.OnDestroy (282), RemoveDataModule (199)

2.40.7 TDataModule.AfterConstruction

Synopsis: Overrides standard TObject (185) behaviour.

Declaration:
```
procedure AfterConstruction;  Override
```

Visibility: public

Description: `AfterConstruction` calls the OnCreate (281) handler if the OldCreateOrder (282) property is `False`.

See also: TDataModule.OldCreateOrder (282), TDataModule.OnCreate (281)

2.40.8 TDataModule.BeforeDestruction

Synopsis:

Declaration:
```
procedure BeforeDestruction;  Override
```

Visibility: public

Description: `BeforeDestruction` calls the OnDestroy (281) handler if the OldCreateOrder (282) property is `False`.

See also: TDataModule.OldCreateOrder (282), TDataModule.OnDestroy (282)

2.40.9 TDataModule.DesignOffset

Synopsis: Position property needed for manipulation in an IDE.

Declaration: `Property DesignOffset : TPoint`

Visibility: public

Access: Read,Write

Description: `DesignOffset` is the position of the datamodule when displayed in an IDE. It is streamed to the form file, and should not be used at run-time.

See also: TDataModule.DesignSize (281)

2.40.10 TDataModule.DesignSize

Synopsis: Size property needed for manipulation in an IDE.

Declaration: `Property DesignSize : TPoint`

Visibility: public

Access: Read,Write

Description: `DesignSize` is the size of the datamodule when displayed in an IDE. It is streamed to the form file, and should not be used at run-time.

See also: TDataModule.DesignOffset (281)

2.40.11 TDataModule.OnCreate

Synopsis: Event handler, called when the datamodule is created.

Declaration: `Property OnCreate : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: The `OnCreate` event is triggered when the datamodule is created and streamed. The exact moment of triggering is dependent on the value of the OldCreateOrder (282) property.

See also: TDataModule.Create (279), TDataModule.CreateNew (280), TDataModule.OldCreateOrder (282)

2.40.12 TDataModule.OnDestroy

Synopsis: Event handler, called when the datamodule is destroyed.

Declaration: `Property OnDestroy : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: The `OnDestroy` event is triggered when the datamodule is destroyed The exact moment of triggering is dependent on the value of the OldCreateOrder (282) property.

See also: TDataModule.Destroy (280), TDataModule.OnCreate (281), TDataModule.Create (279), TDataModule.CreateNew (280), TDataModule.OldCreateOrder (282)

2.40.13 TDataModule.OldCreateOrder

Synopsis: Determines when `OnCreate` and `OnDestroy` are triggered.

Declaration: `Property OldCreateOrder : Boolean`

Visibility: published

Access: Read,Write

Description: `OldCreateOrder` determines when exactly the OnCreate (281) and OnDestroy (282) event handlers are called:

See also: TDataModule.OnDestroy (282), TDataModule.OnCreate (281), TDataModule.Destroy (280), TDataModule.Create (279), TDataModule.CreateNew (280), TDataModule.OldCreateOrder (282)

2.41 TFiler

2.41.1 Description

Class responsible for streaming of components.

2.41.2 Method overview

Page	Property	Description
283	DefineBinaryProperty	
283	DefineProperty	

2.41.3 Property overview

Page	Property	Access	Description
284	Ancestor	rw	Ancestor component from which an inherited component is streamed.
284	IgnoreChildren	rw	Determines whether children will be streamed as well.
283	LookupRoot	r	Component used to look up ancestor components.
283	Root	rw	The root component is the initial component which is being streamed.

2.41.4 TFiler.DefineProperty

Synopsis:

Declaration:
```
procedure DefineProperty(const Name: String;ReadData: TReaderProc;
                         WriteData: TWriterProc;HasData: Boolean)
                         ;  Virtual;  Abstract
```

Visibility: public

Description:

2.41.5 TFiler.DefineBinaryProperty

Synopsis:

Declaration:
```
procedure DefineBinaryProperty(const Name: String;ReadData: TStreamProc;
                               WriteData: TStreamProc;HasData: Boolean)
                               ;  Virtual;  Abstract
```

Visibility: public

Description:

2.41.6 TFiler.Root

Synopsis: The root component is the initial component which is being streamed.

Declaration: `Property Root : TComponent`

Visibility: public

Access: Read,Write

Description: The streaming process will stream a component and all the components which it owns. The `Root` component is the component which is initially streamed.

See also: TFiler.LookupRoot (283)

2.41.7 TFiler.LookupRoot

Synopsis: Component used to look up ancestor components.

Declaration: `Property LookupRoot : TComponent`

Visibility: public

Access: Read

Description: When comparing inherited component's values against parent values, the values are compared with the component in `LookupRoot`. Initially, it is set to Root (283).

See also: TFiler.Root (283)

2.41.8 TFiler.Ancestor

Synopsis: Ancestor component from which an inherited component is streamed.

Declaration: `Property Ancestor : TPersistent`

Visibility: public

Access: Read,Write

Description: When streaming a component, this is the parent component. Only properties that differ from the parent's property value will be streamed.

See also: TFiler.Root (283), TFiler.LookupRoot (283)

2.41.9 TFiler.IgnoreChildren

Synopsis: Determines whether children will be streamed as well.

Declaration: `Property IgnoreChildren : Boolean`

Visibility: public

Access: Read,Write

Description: By default, all children (i.e. owned objects) will also be streamed when streaming a component. This property can be used to prevent owned objects from being streamed.

2.42 TFileStream

2.42.1 Description

`TFileStream` is a TStream (332) descdendent that stores or reads it's data from a named file in the filesystem of the operating system.

To this end, it overrides some of the methods in `TStream` and implements them for the case of files on disk, and it adds the FileName (285) property to the list of public properties.

2.42.2 Method overview

Page	Property	Description
284	Create	Creates a file stream.
285	Destroy	Destroys the file stream.

2.42.3 Property overview

Page	Property	Access	Description
285	FileName	r	The filename of the stream.

2.42.4 TFileStream.Create

Synopsis: Creates a file stream.

Declaration:
```
constructor Create(const AFileName: String;Mode: Word)
constructor Create(const AFileName: String;Mode: Word;Rights: Cardinal)
```

Visibility: public

Description: `Create` creates a new instance of a `TFileStream` class. It opens the file `AFileName` with mode `Mode`, which can have one of the following values:

Table 2.18:

fmCreate	TFileStream.Create (284) creates a new file if needed.
fmOpenRead	TFileStream.Create (284) opens a file with read-only access.
fmOpenWrite	TFileStream.Create (284) opens a file with write-only access.
fmOpenReadWrite	TFileStream.Create (284) opens a file with read-write access.

After the file has been opened in the requested mode and a handle has been obtained from the operating system, the inherited constructor is called.

Errors: If the file could not be opened in the requested mode, an EFOpenError (215) exception is raised.

See also: TStream (332), TFileStream.FileName (285), THandleStream.Create (293)

2.42.5 TFileStream.Destroy

Synopsis: Destroys the file stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` closes the file (causing possible buffered data to be written to disk) and then calls the inherited destructor.

Do not call `destroy` directly, instead call the `Free` method. Destroy does not check whether `Self` is nil, while `Free` does.

See also: TFileStream.Create (284)

2.42.6 TFileStream.FileName

Synopsis: The filename of the stream.

Declaration: `Property FileName : String`

Visibility: public

Access: Read

Description: `FileName` is the name of the file that the stream reads from or writes to. It is the name as passed in the constructor of the stream; it cannot be changed. To write to another file, the stream must be freed and created again with the new filename.

See also: TFileStream.Create (284)

2.43 TFPList

2.43.1 Description

`TFPList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed. Contrary to TList (300), `TFPList` has no notification mechanism. If no notification mechanism is used, it is better to use `TFPList` instead of `TList`, as the performance of `TFPList` is much higher.

To manage collections of strings, it is better to use a TStrings (349) descendent such as TStringList (345). To manage general objects, a TCollection (260) class exists, from which a descendent can be made to manage collections of various kinds.

2.43.2 Method overview

Page	Property	Description
287	Add	Adds a new pointer to the list.
287	AddList	Add all pointers from another list
290	Assign	Assigns all items of a list to this list.
287	Clear	Clears the pointer list.
287	Delete	Removes a pointer from the list.
286	Destroy	Destroys the list and releases the memory used to store the list elements.
288	Error	Raises an EListError (215) exception.
288	Exchange	Exchanges two pointers in the list.
288	Expand	Increases the capacity of the list if needed.
288	Extract	Remove the first occurrence of a pointer from the list.
289	First	Returns the first non-nil pointer in the list.
291	ForEachCall	Call a procedure or method for each pointer in the list.
289	IndexOf	Returns the index of a given pointer.
289	Insert	Inserts a new pointer in the list at a given position.
289	Last	Returns the last non-nil pointer in the list.
290	Move	Moves a pointer from one position in the list to another.
290	Pack	Removes `Nil` pointers from the list and frees unused memory.
290	Remove	Removes a value from the list.
291	Sort	Sorts the pointers in the list.

2.43.3 Property overview

Page	Property	Access	Description
291	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
292	Count	rw	Current number of pointers in the list.
292	Items	rw	Provides access to the pointers in the list.
292	List	r	Memory array where pointers are stored.

2.43.4 TFPList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any meomory they point to must be explicitly released before calling the destructor.

2.43.5 TFPList.AddList

Synopsis: Add all pointers from another list

Declaration: `procedure AddList(AList: TFPList)`

Visibility: public

Description: `AddList` adds all pointers from `AList` to the list. If a pointer is already present, it is added a second time.

See also: TFPList.Assign (290), TList.AddList (301)

2.43.6 TFPList.Add

Synopsis: Adds a new pointer to the list.

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: public

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the Grow (286) method.

To insert a pointer at a certain position in the list, use the Insert (289) method instead.

See also: TFPList.Delete (287), TFPList.Grow (286), TFPList.Insert (289)

2.43.7 TFPList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: TFPList.Destroy (286)

2.43.8 TFPList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

2.43.9 TFPList.Error

Synopsis: Raises an EListError (215) exception.

Declaration: `procedure Error(const Msg: String;Data: PtrInt)`

Visibility: public

Description: `Error` raises an EListError (215) exception, with a message formatted with `Msg` and `Data`.

2.43.10 TFPList.Exchange

Synopsis: Exchanges two pointers in the list.

Declaration: `procedure Exchange(Index1: Integer;Index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be withing the current range of the list, or an EListError (215) exception will be raised.

2.43.11 TFPList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TFPList`

Visibility: public

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

1. If the capacity is less than 3, the capacity is increased with 4.
2. If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
3. If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: TFPList.Capacity (291)

2.43.12 TFPList.Extract

Synopsis: Remove the first occurrence of a pointer from the list.

Declaration: `function Extract(Item: Pointer) : Pointer`

Visibility: public

Description: `Extract` searches for the first occurrence of `Item` in the list and deletes it from the list. If `Item` was found, it's value is returned. If `Item` was not found, `Nil` is returned.

See also: TFPList.Delete (287)

2.43.13 TFPList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: public

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: TFPList.Last (289)

2.43.14 TFPList.IndexOf

Synopsis: Returns the index of a given pointer.

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

2.43.15 TFPList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer;Item: Pointer)`

Visibility: public

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a EListError (215) exception is raised.

See also: TFPList.Add (287), TFPList.Delete (287)

2.43.16 TFPList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: public

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: TFPList.First (289)

2.43.17 TFPList.Move

Synopsis: Moves a pointer from one position in the list to another.

Declaration: `procedure Move(CurIndex: Integer;NewIndex: Integer)`

Visibility: public

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

If `CurIndex` or `Newindex` are not inside the valid range of indices, an EListError (215) exception is raised.

See also: TFPList.Exchange (288)

2.43.18 TFPList.Assign

Synopsis: Assigns all items of a list to this list.

Declaration: `procedure Assign(ListA: TFPList;AOperator: TListAssignOp;ListB: TFPList)`

Visibility: public

Description: `Assign` clears the list and adds all pointers in `Obj` to the list.

See also: TFPList.Add (287), TFPList.Clear (287)

2.43.19 TFPList.Remove

Synopsis: Removes a value from the list.

Declaration: `function Remove(Item: Pointer) : Integer`

Visibility: public

Description: `Remove` searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: TFPList.Delete (287), TFPList.IndexOf (289), TFPList.Insert (289)

2.43.20 TFPList.Pack

Synopsis: Removes `Nil` pointers from the list and frees unused memory.

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded nil pointers in it.

See also: TFPList.Clear (287)

2.43.21 TFPList.Sort

Synopsis: Sorts the pointers in the list.

Declaration:
```
procedure Sort(Compare: TListSortCompare)
```

Visibility: public

Description: `Sort>` sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.
- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

2.43.22 TFPList.ForEachCall

Synopsis: Call a procedure or method for each pointer in the list.

Declaration:
```
procedure ForEachCall(proc2call: TListCallback;arg: pointer)
procedure ForEachCall(proc2call: TListStaticCallback;arg: pointer)
```

Visibility: public

Description: `ForEachCall` iterates over all pointers in the list and calls `proc2call`, passing it the pointer and the additional `arg` data pointer. `Proc2Call` can be a method or a static procedure.

Errors: None.

See also: TListStaticCallback (193), TListCallback (193)

2.43.23 TFPList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using add (287) or insert (289), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: TFPList.SetCapacity (286), TFPList.Count (292)

2.43.24 TFPList.Count

Synopsis: Current number of pointers in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

2.43.25 TFPList.Items

Synopsis: Provides access to the pointers in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` is used to access the pointers in the list. It is the default property of the `TFPList` class, so it can be omitted.

The list is zero-based, so Index must be in the range `0` to `Count-1`.

2.43.26 TFPList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: public

Access: Read

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pinters to another location.

2.44 THandleStream

2.44.1 Description

`THandleStream` is an abstract descendent of the TStream (332) class that provides methods for a stream to handle all reading and writing to and from a handle, provided by the underlying OS. To this end, it overrides the Read (293) and Write (293) methods of TStream.

Remark:

- `ThandleStream` does not obtain a handle from the OS by itself, it just handles reading and writing to such a handle by wrapping the system calls for reading and writing; Descendent classes should obtain a handle from the OS by themselves and pass it on in the inherited constructor.
- Contrary to Delphi, no seek is implemented for `THandleStream`, since pipes and sockets do not support this. The seek is implemented in descendent methods that support it.

2.44.2 Method overview

Page	Property	Description
293	Create	Create a handlestream from an OS Handle.
293	Read	Overrides standard read method.
294	Seek	Overrides the Seek method.
293	Write	Overrides standard write method.

2.44.3 Property overview

Page	Property	Access	Description
294	Handle	r	The OS handle of the stream.

2.44.4 THandleStream.Create

Synopsis: Create a handlestream from an OS Handle.

Declaration: `constructor Create(AHandle: Integer)`

Visibility: public

Description: `Create` creates a new instance of a `THandleStream` class. It stores `AHandle` in an internal variable and then calls the inherited constructor.

See also: TStream (332)

2.44.5 THandleStream.Read

Synopsis: Overrides standard read method.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` overrides the Read (333) method of `TStream`. It uses the Handle (294) property to read the `Count` bytes into `Buffer`

If no error occurs while reading, the number of bytes actually read will be returned.

Errors: If the operating system reports an error while reading from the handle, -1 is returned.

See also: TStream.Read (333), THandleStream.Write (293), THandleStream.Handle (294)

2.44.6 THandleStream.Write

Synopsis: Overrides standard write method.

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` overrides the Write (333) method of `TStream`. It uses the Handle (294) property to write the `Count` bytes from `Buffer`.

If no error occurs while writing, the number of bytes actually written will be returned.

Errors: If the operating system reports an error while writing to the handle, 0 is returned.

See also: TStream.Read (333), THandleStream.Write (293), THandleStream.Handle (294)

2.44.7 THandleStream.Seek

Synopsis: Overrides the Seek method.

Declaration:
```
function Seek(const Offset: Int64;Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `seek` uses the FileSeek (1417) method to position the stream on the desired position. Note that handle stream descendents (notably pipes) can override the method to prevent the seek.

2.44.8 THandleStream.Handle

Synopsis: The OS handle of the stream.

Declaration:
```
Property Handle : Integer
```

Visibility: public

Access: Read

Description: `Handle` represents the Operating system handle to which reading and writing is done. The handle can be read only, i.e. it cannot be set after the `THandlestream` instance was created. It should be passed to the constructor THandleStream.Create (293)

See also: THandleStream (292), THandleStream.Create (293)

2.45 TInterfacedPersistent

2.45.1 Description

`TInterfacedPersistent` is a direct descendent of TPersistent (317) which implements the #rtl.system.IInterface (1183) interface. In particular, it implements the `QueryInterface` as a public method.

2.45.2 Method overview

Page	Property	Description
295	AfterConstruction	Overrides the standard `AfterConstruction` method.
294	QueryInterface	Implementation of `IInterface.QueryInterface`

2.45.3 TInterfacedPersistent.QueryInterface

Synopsis: Implementation of `IInterface.QueryInterface`

Declaration:
```
function QueryInterface(const IID: TGUID;out Obj) : HResult;  Virtual
```

Visibility: public

Description: `QueryInterface` simply calls `GetInterface` using the specified `IID`, and returns the correct values.

See also: #rtl.system.tobject.GetInterface (1348)

2.45.4 TInterfacedPersistent.AfterConstruction

Synopsis: Overrides the standard `AfterConstruction` method.

Declaration: `procedure AfterConstruction; Override`

Visibility: public

Description: `AfterConstruction` is overridden to do some extra interface housekeeping: a reference to the `IInterface` interface of the owning class is obtained (if it exists).

2.46 TInterfaceList

2.46.1 Description

`TInterfaceList` is a standard implementation of the IInterfaceList (219) interface. It uses a TThreadList (370) instance to store the list of interfaces.

2.46.2 Method overview

Page	Property	Description
297	Add	Add an interface to the list
296	Clear	Removes all interfaces from the list.
295	Create	Create a new instance of `TInterfaceList`
296	Delete	Delete an interface from the list.
296	Destroy	Destroys the list of interfaces
296	Exchange	Exchange 2 interfaces in the list
299	Expand	Expands the list
297	First	Returns the first non-`Nil` element in the list.
297	IndexOf	Returns the index of an interface.
297	Insert	Insert an interface to the list
298	Last	Returns the last non-`Nil` element in the list.
298	Lock	Lock the list
298	Remove	Remove an interface from the list
298	Unlock	UnLocks a locked list

2.46.3 Property overview

Page	Property	Access	Description
299	Capacity	rw	The current capacity of the list.
299	Count	rw	The current number of elements in the list.
299	Items	rw	Array-based access to the list's items.

2.46.4 TInterfaceList.Create

Synopsis: Create a new instance of `TInterfaceList`

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of the `TInterfaceList` class. It sets up the internal structures needed to store the list of interfaces.

See also: TInterfaceList.Destroy (296)

2.46.5 TInterfaceList.Destroy

Synopsis: Destroys the list of interfaces

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` first calls Clear (296) and then frees the `TInterfaceList` instance from memory.

Note that the `Clear` method decreases the reference count of all interfaces.

See also: TInterfaceList.Create (295), TInterfaceList.Clear (296)

2.46.6 TInterfaceList.Clear

Synopsis: Removes all interfaces from the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` is the implementation of the IInterfaceList.Clear (221) method. It removes all interfaces from the list. It does this by setting each element in the list to `Nil`, in this way the reference count of each interface in the list is descreased.

See also: IInterfaceList.Clear (221), TInterfaceList.Add (297), TInterfaceList.Destroy (296), TList.Clear (301), TFPList.Clear (287)

2.46.7 TInterfaceList.Delete

Synopsis: Delete an interface from the list.

Declaration: `procedure Delete(index: Integer)`

Visibility: public

Description: `Delete` is the implementation of the IInterfaceList.Delete (221) method. It clears the slot first and then removes the element from the list.

See also: IInterfaceList.Delete (221), TInterfaceList.Remove (298), TInterfaceList.Add (297), TList.Delete (302), TFPList.Delete (287)

2.46.8 TInterfaceList.Exchange

Synopsis: Exchange 2 interfaces in the list

Declaration: `procedure Exchange(index1: Integer;index2: Integer)`

Visibility: public

Description: `Exchange` is the implementation of the IInterfaceList.Exchange (221) method. It exchanges the position of 2 interfaces in the list.

See also: IInterfaceList.Exchange (221), TInterfaceList.Delete (296), TInterfaceList.Add (297), TList.Exchange (302), TFPList.Exchange (288)

2.46.9 TInterfaceList.First

Synopsis: Returns the first non-`Nil` element in the list.

Declaration: `function First : IUnknown`

Visibility: public

Description: `First` is the implementation of the IInterfaceList.First (222) method. It returns the first non-`Nil` element from the list.

See also: IInterfaceList.First (222), TList.First (303)

2.46.10 TInterfaceList.IndexOf

Synopsis: Returns the index of an interface.

Declaration: `function IndexOf(item: IUnknown) : Integer`

Visibility: public

Description: `IndexOf` is the implementation of the IInterfaceList.IndexOf (222) method. It returns the zero-based index in the list of the indicated interface, or -1 if the index is not in the list.

See also: IInterfaceList.IndexOf (222), TList.IndexOf (303)

2.46.11 TInterfaceList.Add

Synopsis: Add an interface to the list

Declaration: `function Add(item: IUnknown) : Integer`

Visibility: public

Description: `Add` is the implementation of the IInterfaceList.Add (222) method. It adds an interface to the list, and returns the location of the new element in the list. This operation will increment the reference count of the interface.

See also: IInterfaceList.Add (222), TInterfaceList.Delete (296), TInterfaceList.Insert (297), TList.Add (301), TFPList.Add (287)

2.46.12 TInterfaceList.Insert

Synopsis: Insert an interface to the list

Declaration: `procedure Insert(i: Integer;item: IUnknown)`

Visibility: public

Description: `Insert` is the implementation of the IInterfaceList.Insert (222) method. It inserts an interface in the list at the indicated position. This operation will increment the reference count of the interface.

See also: IInterfaceList.Insert (222), TInterfaceList.Delete (296), TInterfaceList.Add (297), TList.Insert (303), TFPList.Insert (289)

2.46.13 TInterfaceList.Last

Synopsis: Returns the last non-`Nil` element in the list.

Declaration: `function Last : IUnknown`

Visibility: public

Description: `Last` is the implementation of the IInterfaceList.Last (222) method. It returns the last non-`Nil` element from the list.

See also: IInterfaceList.Last (222), TInterfaceList.First (297), TList.Last (304), TFPList.Last (289)

2.46.14 TInterfaceList.Remove

Synopsis: Remove an interface from the list

Declaration: `function Remove(item: IUnknown) : Integer`

Visibility: public

Description: `Remove` is the implementation of the IInterfaceList.Remove (223) method. It removes the first occurrence of the interface from the list.

See also: IInterfaceList.Remove (223), TInterfaceList.Delete (296), TInterfaceList.IndexOf (297), TList.Remove (304), TFList.Remove (185)

2.46.15 TInterfaceList.Lock

Synopsis: Lock the list

Declaration: `procedure Lock`

Visibility: public

Description: `Lock` locks the list. It is the implementation of the IInterfaceList.Lock (223) method. It limits access to the list to the current thread.

See also: IInterfaceList.Lock (223), TInterfaceList.UnLock (298), TThreadList.LockList (371)

2.46.16 TInterfaceList.Unlock

Synopsis: UnLocks a locked list

Declaration: `procedure Unlock`

Visibility: public

Description: `UnLock` unlocks the list. It is the implementation of the IInterfaceList.UnLock (223) method. After a call to unlock, the current thread releases the list for manipulation by other threads.

See also: IInterfaceList.UnLock (223), TInterfaceList.Lock (298), TThreadList.UnLockList (371)

2.46.17 TInterfaceList.Expand

Synopsis: Expands the list

Declaration: `function Expand : TInterfaceList`

Visibility: public

Description: `Expand` calls the expand method from the internally used list. It returns itself.

See also: TList.Expand (302)

2.46.18 TInterfaceList.Capacity

Synopsis: The current capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of elements that the list can contain without needing to allocate more memory.

See also: IInterfaceList.Capacity (223), TInterfaceList.Count (299), TList.Capacity (305), TFPList.Capacity (291)

2.46.19 TInterfaceList.Count

Synopsis: The current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the number of elements in the list. This can include `Nil` elements. Note that the elements are zero-based, and thus are indexed from 0 to `Count-1`.

See also: IInterfaceList.Count (224), TInterfaceList.Items (299), TInterfaceList.Capacity (299), TList.Count (306), TFPList.Count (292)

2.46.20 TInterfaceList.Items

Synopsis: Array-based access to the list's items.

Declaration: `Property Items[Index: Integer]: IUnknown; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the elements in the list. Note that the elements are zero-based, and thus are indexed from 0 to `Count-1`. The items are read-write. It is not possible to add elements to the list by accessing an element with index larger or equal to Count (299).

See also: IInterfaceList.Items (224), TInterfaceList.Count (299), TList.Items (306), TFPList.Items (292)

2.47 TList

2.47.1 Description

`TList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed. It has an event notification mechanism which allows to notify of list changes. This slows down some of `TList` mechanisms, and if no notification is used, TFPList (286) may be used instead.

To manage collections of strings, it is better to use a TStrings (349) descendent such as TStringList (345). To manage general objects, a TCollection (260) class exists, from which a descendent can be made to manage collections of various kinds.

2.47.2 Method overview

Page	Property	Description
301	Add	Adds a new pointer to the list.
301	AddList	Add all pointers from another list
304	Assign	Copy the contents of another list.
301	Clear	Clears the pointer list.
300	Create	Class to manage collections of pointers.
302	Delete	Removes a pointer from the list.
301	Destroy	Destroys the list and releases the memory used to store the list elements.
302	Error	Raises an EListError (215) exception.
302	Exchange	Exchanges two pointers in the list.
302	Expand	Increases the capacity of the list if needed.
303	Extract	Remove the first occurrence of a pointer from the list.
303	First	Returns the first non-nil pointer in the list.
303	IndexOf	Returns the index of a given pointer.
303	Insert	Inserts a new pointer in the list at a given position.
304	Last	Returns the last non-nil pointer in the list.
304	Move	Moves a pointer from one position in the list to another.
305	Pack	Removes `Nil` pointers from the list and frees unused memory.
304	Remove	Removes a value from the list.
305	Sort	Sorts the pointers in the list.

2.47.3 Property overview

Page	Property	Access	Description
305	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
306	Count	rw	Current number of pointers in the list.
306	Items	rw	Provides access to the pointers in the list.
306	List	r	Memory array where pointers are stored.

2.47.4 TList.Create

Synopsis: Class to manage collections of pointers.

Declaration: `constructor Create`

Visibility: public

Description: `TList.Create` creates a new instance of `TList`. It clears the list and prepares it for use.

See also: TList (300), TList.Destroy (301)

2.47.5 TList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any meomory they point to must be explicitly released before calling the destructor.

2.47.6 TList.AddList

Synopsis: Add all pointers from another list

Declaration: `procedure AddList(AList: TList)`

Visibility: public

Description: `AddList` adds all pointers from `AList` to the list. If a pointer is already present, it is added a second time.

See also: TList.Assign (304), TFPList.AddList (287)

2.47.7 TList.Add

Synopsis: Adds a new pointer to the list.

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: public

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the Grow (300) method.

To insert a pointer at a certain position in the list, use the Insert (303) method instead.

See also: TList.Delete (302), TList.Grow (300), TList.Insert (303)

2.47.8 TList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear;  Virtual`

Visibility: public

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: TList.Destroy (301)

2.47.9 TList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

2.47.10 TList.Error

Synopsis: Raises an EListError (215) exception.

Declaration: `procedure Error(const Msg: String;Data: PtrInt);  Virtual`

Visibility: public

Description: `Error` raises an EListError (215) exception, with a message formatted with `Msg` and `Data`.

2.47.11 TList.Exchange

Synopsis: Exchanges two pointers in the list.

Declaration: `procedure Exchange(Index1: Integer;Index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be withing the current range of the list, or an EListError (215) exception will be raised.

2.47.12 TList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TList`

Visibility: public

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

1. If the capacity is less than 3, the capacity is increased with 4.
2. If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
3. If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: TList.Capacity (305)

2.47.13 TList.Extract

Synopsis: Remove the first occurrence of a pointer from the list.

Declaration: `function Extract(item: Pointer) : Pointer`

Visibility: public

Description: `Extract` searched for an occurence of `item`, and if a match is found, the match is deleted from the list. If no match is found, nothing is deleted. If `Item` was found, the result is `Item`. If `Item` was not found, the result is `Nil`. A `lnExtracted` notification event is triggered if an element is extracted from the list. Note that a `lnDeleted` event will also occur.

See also: TList.Delete (302), TList.IndexOf (303), TList.Remove (304)

2.47.14 TList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: public

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: TList.Last (304)

2.47.15 TList.IndexOf

Synopsis: Returns the index of a given pointer.

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

2.47.16 TList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer;Item: Pointer)`

Visibility: public

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a EListError (215) exception is raised.

See also: TList.Add (301), Tlist.Delete (302)

2.47.17 TList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: public

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: TList.First (303)

2.47.18 TList.Move

Synopsis: Moves a pointer from one position in the list to another.

Declaration: `procedure Move(CurIndex: Integer;NewIndex: Integer)`

Visibility: public

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

If `CurIndex` or `Newindex` are not inside the valid range of indices, an EListError (215) exception is raised.

See also: TList.Exchange (302)

2.47.19 TList.Assign

Synopsis: Copy the contents of another list.

Declaration: `procedure Assign(ListA: TList;AOperator: TListAssignOp;ListB: TList)`

Visibility: public

Description: `Assign` copies the pointers of the `Obj` list to the list. The list is cleared prior to copying.

See also: TList.Clear (301)

2.47.20 TList.Remove

Synopsis: Removes a value from the list.

Declaration: `function Remove(Item: Pointer) : Integer`

Visibility: public

Description: `Remove` searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: TList.Delete (302), TList.IndexOf (303), Tlist.Insert (303)

2.47.21 TList.Pack

Synopsis: Removes `Nil` pointers from the list and frees unused memory.

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded nil pointers in it.

See also: TList.Clear (301)

2.47.22 TList.Sort

Synopsis: Sorts the pointers in the list.

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: public

Description: `Sort>` sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.
- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

2.47.23 TList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using add (301) or insert (303), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: TList.SetCapacity (300), TList.Count (306)

2.47.24 TList.Count

Synopsis: Current number of pointers in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

2.47.25 TList.Items

Synopsis: Provides access to the pointers in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` is used to access the pointers in the list. It is the default property of the `TList` class, so it can be omitted.

The list is zero-based, so Index must be in the range `0` to `Count-1`.

2.47.26 TList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: public

Access: Read

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pinters to another location.

2.48 TMemoryStream

2.48.1 Description

`TMemoryStream` is a TStream (332) descendent that stores it's data in memory. It descends directly from TCustomMemoryStream (276) and implements the necessary to allocate and de-allocate memory directly from the heap. It implements the Write (308) method which is missing in `TCustomMemoryStream`.

`TMemoryStream` also introduces methods to load the contents of another stream or a file into the memory stream.

It is not necessary to do any memory management manually, as the stream will allocate or de-allocate memory as needed. When the stream is freed, all allocated memory will be freed as well.

2.48.2 Method overview

Page	Property	Description
307	Clear	Zeroes the position, capacity and size of the stream.
307	Destroy	Frees any allocated memory and destroys the memory stream.
308	LoadFromFile	Loads the contents of a file into memory.
307	LoadFromStream	Loads the contents of a stream into memory.
308	SetSize	Sets the size for the memory stream.
308	Write	Writes data to the stream's memory.

2.48.3 TMemoryStream.Destroy

Synopsis: Frees any allocated memory and destroys the memory stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Free` clears the memory stream, thus in effect freeing any memory allocated for it, and then frees the memory stream.

2.48.4 TMemoryStream.Clear

Synopsis: Zeroes the position, capacity and size of the stream.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` sets the position and size to 0, and sets the capacity of the stream to 0, thus freeing all memory allocated for the stream.

See also: TStream.Size (340), TStream.Position (340), TCustomMemoryStream.Memory (278)

2.48.5 TMemoryStream.LoadFromStream

Synopsis: Loads the contents of a stream into memory.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the memorybuffer of the stream. Any previous contents of the memory stream are overwritten. Memory is allocated as needed.

Remark: The `LoadFromStream` uses the Size (340) property of `Stream` to determine how much memory must be allocated. Some streams do not allow the stream size to be determined, so care must be taken when using this method.

This method will work much faster than the use of the TStream.CopyFrom (335) method:

```
Seek(0,soFromBeginning);
  CopyFrom(Stream,Stream.Size);
```

because the `CopyFrom` method copies the contents in blocks, while `LoadFromStream` reads the contents of the stream as one big block.

Errors: If an error occurs when reading from the stream, an EStreamError (216) may occur.

See also: TStream.CopyFrom (335), TMemoryStream.LoadFromFile (308)

2.48.6 TMemoryStream.LoadFromFile

Synopsis: Loads the contents of a file into memory.

Declaration: `procedure LoadFromFile(const FileName: String)`

Visibility: public

Description: `LoadFromFile` loads the contents of the file with name `FileName` into the memory stream. The current contents of the memory stream is replaced by the contents of the file. Memory is allocated as needed.

The `LoadFromFile` method simply creates a filestream and then calls the TMemoryStream.LoadFromStream (307) method.

See also: TMemoryStream.LoadFromStream (307)

2.48.7 TMemoryStream.SetSize

Synopsis: Sets the size for the memory stream.

Declaration: `procedure SetSize(NewSize: LongInt);  Override`

Visibility: public

Description: `SetSize` sets the size of the memory stream to `NewSize`. This will set the capacity of the stream to `NewSize` and correct the current position in the stream when needed.

See also: TStream.Position (340), TStream.Size (340)

2.48.8 TMemoryStream.Write

Synopsis: Writes data to the stream's memory.

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the stream's memory, starting at the current position in the stream. If more memory is needed than currently allocated, more memory will be allocated. Any contents in the memory stream at the current position will be overwritten. The function returns the number of bytes actually written (which should under normal circumstances always equal `Count`).

This method overrides the TStream.Write (333) method.

Errors: If no more memory could be allocated, then an exception will be raised.

See also: TCustomMemoryStream.Read (277)

2.49 TOwnedCollection

2.49.1 Description

`TOwnedCollection` automatically maintains owner information, so it can be displayed in an IDE. Collections that should be displayed in an IDE should descend from `TOwnedCollection` or must implement a `GetOwner` function.

2.49.2 Method overview

Page	Property	Description
309	Create	Create a new `TOwnerCollection` instance.

2.49.3 TOwnedCollection.Create

Synopsis: Create a new `TOwnerCollection` instance.

Declaration:
```
constructor Create(AOwner: TPersistent;AItemClass: TCollectionItemClass)
```

Visibility: public

Description: `Create` creates a new instance of `TOwnedCollection` and stores the `AOwner` references. It will the value returned in the TCollection.Owner (261) property of the collection. The `ItemClass` class reference is passed on to the inherited constructor, and will be used to create new instances in the Insert (263) and Add (262) methods.

See also: TCollection.Create (261), TCollection.Owner (261)

2.50 TOwnerStream

2.50.1 Description

`TOwnerStream` can be used when creating stream chains such as when using encryption and compression streams. It keeps a reference to the source stream and will automatically free the source stream when ready (if the SourceOwner (310) property is set to `True`).

2.50.2 Method overview

Page	Property	Description
309	Create	Create a new instance of `TOwnerStream`.
310	Destroy	Destroys the `TOwnerStream` instance and the source stream.

2.50.3 Property overview

Page	Property	Access	Description
310	Source	r	Reference to the source stream.
310	SourceOwner	rw	Indicates whether the ownerstream owns it's source

2.50.4 TOwnerStream.Create

Synopsis: Create a new instance of `TOwnerStream`.

Declaration:
```
constructor Create(ASource: TStream)
```

Visibility: public

Description: `Create` instantiates a new instance of `TOwnerStream` and stores the reference to `AStream`. If `SourceOwner` is `True`, the soure stream will also be freed when the instance is destroyed.

See also: TOwnerStream.Destroy (310), TOwnerStream.Source (310), TOwnerStream.SourceOwner (310)

2.50.5 TOwnerStream.Destroy

Synopsis: Destroys the `TOwnerStream` instance and the source stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the source stream if the `SourceOwner` property is `True`.

Errors:

See also: TOwnerStream.Create (309), TOwnerStream.Source (310), TOwnerStream.SourceOwner (310)

2.50.6 TOwnerStream.Source

Synopsis: Reference to the source stream.

Declaration: `Property Source : TStream`

Visibility: public

Access: Read

Description: `Source` is the source stream. It should be uses by descendent streams to access the source stream to read from or write to.

Do not free the `Source` reference directly. Either the owner stream instance should free the source stream or

See also: TOwnerStream.Create (309)

2.50.7 TOwnerStream.SourceOwner

Synopsis: Indicates whether the ownerstream owns it's source

Declaration: `Property SourceOwner : Boolean`

Visibility: public

Access: Read,Write

Description: `SourceOwner` indicates whether the `TOwnerStream` owns it's `Source` stream or not. If this property is `True` then the `Source` stream is freed when the `TOwnerStream` instance is freed.

See also: TOwnerStream.Source (310), TOwnerStream.Destroy (310)

2.51 TParser

2.51.1 Description

This class breaks a stream of text data in tokens. Its primary use is to help reading the contents of a form file (usually a file with dfm, xfm or lfm extension), and for this reason it isn't suitable to be used as a general parser.

The parser is always positioned on a certain token, whose type is stored in the Token (316) property. Various methods are provided to obtain the token value in the desired format.

To advance to the next token, invoke NextToken (313) method.

2.51.2 Method overview

Page	Property	Description
312	CheckToken	Checks whether the token if of the given type.
312	CheckTokenSymbol	Checks whether the token equals the given symbol
311	Create	Creates a new parser instance.
311	Destroy	Destroys the parser instance.
312	Error	Raises an EParserError (216) exception with the given message
312	ErrorFmt	Raises an EParserError (216) exception and formats the message.
312	ErrorStr	Raises an EParserError (216) exception with the given message
313	HexToBinary	Writes hexadecimal data to a stream.
313	NextToken	Reads the next token and returns its type.
313	SourcePos	Returns the current position in the stream.
314	TokenComponentIdent	Returns the path of a subcomponent starting from the current token.
314	TokenFloat	Returns the current token as a float.
314	TokenInt	Returns the current token as an integer.
315	TokenString	Returns the current token as a string.
315	TokenSymbolIs	Returns `True` if the token equals the given symbol.
315	TokenWideString	Returns the current token as a widestring

2.51.3 Property overview

Page	Property	Access	Description
316	FloatType	r	The type of a float token.
316	SourceLine	r	Current source line number.
316	Token	r	The type of the current token.

2.51.4 TParser.Create

Synopsis: Creates a new parser instance.

Declaration:
```
constructor Create(Stream: TStream)
```

Visibility: public

Description: `Create` creates a new `TParser` instance, using `Stream` as the stream to read data from, and reads the first token from the stream.

Errors: If an error occours while parsing the first token, an EParserError (216) exception is raised.

See also: TParser.NextToken (313), TParser.Token (316)

2.51.5 TParser.Destroy

Synopsis: Destroys the parser instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroys the parser instance.

Errors: None.

2.51.6 TParser.CheckToken

Synopsis: Checks whether the token if of the given type.

Declaration: `procedure CheckToken(T: Char)`

Visibility: public

Description: Checks whether the token if of the given type.

Errors: If current token isn't of type `T`, an EParserError (216) exception is raised.

See also: TParser.Token (316)

2.51.7 TParser.CheckTokenSymbol

Synopsis: Checks whether the token equals the given symbol

Declaration: `procedure CheckTokenSymbol(const S: String)`

Visibility: public

Description: `CheckTokenSymbol` performs a case-insensitive comparison of current token value with `S`. Current token must be of type toSymbol (187), otherwise an EParserError (216) exception is raised.

Errors: If the comparison fails, or current token isn't a symbol, an EParserError (216) exception is raised.

See also: TParser.TokenSymbolIs (315), toSymbol (187)

2.51.8 TParser.Error

Synopsis: Raises an EParserError (216) exception with the given message

Declaration: `procedure Error(const Ident: String)`

Visibility: public

Description: Raises an EParserError (216) exception with the given message

2.51.9 TParser.ErrorFmt

Synopsis: Raises an EParserError (216) exception and formats the message.

Declaration: `procedure ErrorFmt(const Ident: String;const Args: Array of const)`

Visibility: public

Description: Raises an EParserError (216) exception and formats the message.

2.51.10 TParser.ErrorStr

Synopsis: Raises an EParserError (216) exception with the given message

Declaration: `procedure ErrorStr(const Message: String)`

Visibility: public

Description: Raises an EParserError (216) exception with the given message

2.51.11 TParser.HexToBinary

Synopsis: Writes hexadecimal data to a stream.

Declaration: `procedure HexToBinary(Stream: TStream)`

Visibility: public

Description: `HexToBinary` reads a sequence of hexadecimal characters from the input stream and converts them to a sequence of bytes which is written to `Stream`. Each byte is represented by two contiguous hexadecimal characters.

Whitespace is allowed between hexadecimal characters if it doesn't appear between two characters that form the same byte.

`HexToBinary` stops when the first non-hexadecimal and non-whitespace character is found, or the end of the input stream is reached.

Remark: This method begins reading after the current token: that is, current token, even if it's a valid hexadecimal value, isn't included.

Errors: If a single hexadecimal character is found, an EParserError (216) exception is raised.

2.51.12 TParser.NextToken

Synopsis: Reads the next token and returns its type.

Declaration: `function NextToken : Char`

Visibility: public

Description: `NextToken` parses the next token in the stream and returns its type. The type of the token can also be retrieved later reading Token (316) property.

If the end of the stream is reached, toEOF (187) is returned.

For details about token types, see TParser.Token (316)

Errors: If an error occours while parsing the token, an EParserError (216) exception is raised.

See also: TParser.Token (316)

2.51.13 TParser.SourcePos

Synopsis: Returns the current position in the stream.

Declaration: `function SourcePos : LongInt`

Visibility: public

Description: **Remark:** This is not the character position relative to the current source line, but the byte offset from the beginning of the stream.

Errors: None.

See also: TParser.SourceLine (316)

2.51.14 TParser.TokenComponentIdent

Synopsis: Returns the path of a subcomponent starting from the current token.

Declaration: `function TokenComponentIdent : String`

Visibility: public

Description: If current token is toSymbol (187), `TokenComponentIdent` tries to find subcomponent names separated by a dot (`.`). The returned string is the longest subcomponent path found. If there are no subcomponents, current symbol is returned.

Remark: After this method has been called, subsequent calls to TokenString (315) or TokenWideString (315) return the same value returned by `TokenComponentIdent`.

Example

If source stream contains `a.b.c` and `TParser` is positioned on the first token (`a`), this method returns `a.b.c`.

Errors: If Token (316) isn't toSymbol (187), or no valid symbol is found after a dot, an EParserError (216) exception is raised.

See also: TParser.NextToken (313), TParser.Token (316), TParser.TokenString (315), TParser.TokenWideString (315), toSymbol (187)

2.51.15 TParser.TokenFloat

Synopsis: Returns the current token as a float.

Declaration: `function TokenFloat : Extended`

Visibility: public

Description: If current token type is toFloat (187), this method returns the token value as a float.

To specify a negative number, no space must exist between unary minus and number.

Floating point numbers can be postfixed with a character that specifies the floating point type. See FloatType (316) for further information.

Remark: In the input stream the decimal separator, if present, must be a dot (`.`).

Errors: If Token (316) isn't toFloat (187), an EParserError (216) exception is raised.

See also: TParser.FloatType (316), TParser.NextToken (313), TParser.Token (316), toFloat (187)

2.51.16 TParser.TokenInt

Synopsis: Returns the current token as an integer.

Declaration: `function TokenInt : Int64`

Visibility: public

Description: If current token type is toInteger (187), this method returns the token value as an integer.

In the input stream an integer can be an hexadecimal (prefixed by `'$'` character) or decimal number. Decimal numbers can be prefixed by an unary minus: if this is the case, no space must exist between minus and number.

Errors: If Token (316) isn't toInteger (187), an EConvertError (185) exception is raised.

See also: TParser.NextToken (313), TParser.Token (316), toInteger (187)

2.51.17 TParser.TokenString

Synopsis: Returns the current token as a string.

Declaration: `function TokenString : String`

Visibility: public

Description: If current token type is toString (187) or toWString (187), this method returns the contents of the string. That is, enclosing quotes are removed, embedded quotes are unescaped and control strings are converted to the appropriate sequence of characters.

If current token type isn't a string, a string containing the token representation in the input stream is returned, without any conversion: hexadecimal integers are returned with the leading `$`, and floating point suffixes like `s`, `c` or `d` are kept. For tokens whose type isn't a special type, return value of `TokenString` equals Token (316).

Remark: If Token (316) is toWString (187), TokenWideString (315) should be used instead.

Errors: None.

See also: TParser.NextToken (313), TParser.TokenWideString (315), TParser.Token (316), toString (187), toWString (187)

2.51.18 TParser.TokenWideString

Synopsis: Returns the current token as a widestring

Declaration: `function TokenWideString : WideString`

Visibility: public

Description: If current token type is toWString (187), this method returns the contents of the string. That is, enclosing quotes are removed, embedded quotes are unescaped and control strings are converted to the appropriate sequence of characters.

If current token isn't a widestring, `TokenWideString` behaviour is the same as TokenString (315).

Errors: None.

See also: TParser.NextToken (313), TParser.TokenString (315), TParser.Token (316), toWString (187)

2.51.19 TParser.TokenSymbolIs

Synopsis: Returns `True` if the token equals the given symbol.

Declaration: `function TokenSymbolIs(const S: String) : Boolean`

Visibility: public

Description: `TokenSymbolIs` performs a case-insensitive comparison of current token value with `S`.

If current token isn't of type toSymbol (187), or comparison fails, `False` is returned.

Errors: None.

See also: TParser.CheckTokenSymbol (312), TParser.Token (316)

2.51.20 TParser.FloatType

Synopsis: The type of a float token.

Declaration: `Property FloatType : Char`

Visibility: public

Access: Read

Description: Floating point numbers can be postfixed with a character specifying the type of floating point value. When specified, this property holds the character postfixed to the number.

It can be one of the following values:

Table 2.19:

`s` or `S`	Value is a single.
`c` or `C`	Value is a currency.
`d` or `D`	Value is a date.

If Token (316) isn't toFloat (187) or one of the above characters wasn't specified, `FloatType` is the null character (zero).

See also: TParser.NextToken (313), TParser.Token (316), TParser.TokenFloat (314), toFloat (187)

2.51.21 TParser.SourceLine

Synopsis: Current source line number.

Declaration: `Property SourceLine : Integer`

Visibility: public

Access: Read

Description: Current source line number.

See also: TParser.SourcePos (313)

2.51.22 TParser.Token

Synopsis: The type of the current token.

Declaration: `Property Token : Char`

Visibility: public

Access: Read

Description: This property holds the type of the current token. When `Token` isn't one of the special token types (whose value can be retrieved with specific methods) it is the character representing the current token.

Special token types:

To advance to the next token, use NextToken (313) method.

See also: TParser.CheckToken (312), TParser.NextToken (313), TParser.TokenComponentIdent (314), TParser.TokenFloat (314), TParser.TokenInt (314), TParser.TokenString (315), TParser.TokenWideString (315)

Table 2.20:

toEOF (187)	Value returned by TParser.Token (316) when the end of the input stream was reached.
toSymbol (187)	Value returned by TParser.Token (316) when a symbol was found in the input stream.
toString (187)	Value returned by TParser.Token (316) when a string was found in the input stream.
toInteger (187)	Value returned by TParser.Token (316) when an integer was found in the input stream.
toFloat (187)	Value returned by TParser.Token (316) when a floating point value was found in the input stream.
toWString (187)	Value returned by TParser.Token (316) when a widestring was found in the input stream.

2.52 TPersistent

2.52.1 Description

`TPersistent` is the basic class for the streaming system. Since it is compiled in the `{$M+}` state, the compiler generates RTTI (Run-Time Type Information) for it and all classes that descend from it. This information can be used to stream all properties of classes.

It also introduces functionality to assign the contents of 2 classes to each other.

2.52.2 Method overview

Page	Property	Description
317	Assign	Assign the contents of one class to another.
317	Destroy	Destroys the `TPersistent` instance.
318	GetNamePath	Returns a string that can be used to identify the class instance.

2.52.3 TPersistent.Destroy

Synopsis: Destroys the `TPersistent` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` disposes of the persistent object. This method should never be called directly. Instead the `Free` method should be used.

2.52.4 TPersistent.Assign

Synopsis: Assign the contents of one class to another.

Declaration:
```
procedure Assign(Source: TPersistent);  Virtual
```

Visibility: public

Description: `Assign` copies the contents of `Source` to `Self`, if the classes of the destination and source classes are compatible.

The `TPersistent` implementation of `Assign` does nothing but calling the AssignTo (317) method of source. This means that if the destination class does not know how to assign the contents of the source class, the source class instance is asked to assign itself to the destination class. This means that it is necessary to implement only one of the two methods so that two classes can be assiged to one another.

Remark: In general, a statement of the form

```
Destination:=Source;
```

(where `Destination` and `Source` are classes) does not achieve the same as a statement of the form

```
Destination.Assign(Source);
```

After the former statement, both `Source` and `Destination` will point to the same object. The latter statemtent will copy the *contents* of the `Source` class to the `Destination` class.

See also: TPersistent.AssignTo (317)

2.52.5 TPersistent.GetNamePath

Synopsis: Returns a string that can be used to identify the class instance.

Declaration: `function GetNamePath : String;  Virtual`

Visibility: public

Description: `GetNamePath` returns a string that can be used to identify the class instance. This can be used to display a name for this instance in a Object designer.

`GetNamePath` constructs a name by recursively prepending the `Classname` of the `Owner` instance to the `Classname` of this instance, separated by a dot.

See also: TPersistent.GetOwner (317)

2.53 TReader

2.53.1 Description

The `TReader` class is a reader class that implements generic component streaming capabilities, independent of the format of the data in the stream. It uses a driver class TAbstractObjectReader (225) to do the actual reading of data. The interface of the `TReader` class should be identical to the interface in Delphi.

2.53.2 Method overview

Page	Property	Description
320	BeginReferences	Initializes the component referencing mechanism.
321	CheckValue	Raises an exception if the next value in the stream is not of type `Value`
326	CopyValue	Copy a value to a writer.
320	Create	Creates a new reader class
321	DefineBinaryProperty	Reads a user-defined binary property from the stream.
321	DefineProperty	Reads a user-defined property from the stream.
320	Destroy	Destroys a reader class.
321	EndOfList	Returns true if the stream contains an end-of-list marker.
321	EndReferences	Finalizes the component referencing mechanism.
322	FixupReferences	Tries to resolve all unresolved component references.
322	NextValue	Returns the type of the next value.
322	Read	Read raw data from stream
322	ReadBoolean	Reads a boolean from the stream.
322	ReadChar	Reads a character from the stream.
323	ReadCollection	Reads a collection from the stream.
323	ReadComponent	Starts reading a component from the stream.
323	ReadComponents	Starts reading child components from the stream.
324	ReadCurrency	Read a currency value from the stream.
324	ReadDate	Reads a date from the stream
323	ReadFloat	Reads a float from the stream.
324	ReadIdent	Reads an identifier from the stream.
325	ReadInt64	Reads a 64-bit integer from the stream.
324	ReadInteger	Reads an integer from the stream
325	ReadListBegin	Checks for the beginning of a list.
325	ReadListEnd	Checks for the end of a list.
325	ReadRootComponent	Starts reading a root component.
324	ReadSingle	Reads a single-type real from the stream.
325	ReadString	Reads a string from the stream.
326	ReadValue	Reads the next value type from the stream.
323	ReadWideChar	Read widechar from the stream
326	ReadWideString	Read a WideString value from the stream.

2.53.3 Property overview

Page	Property	Access	Description
326	Driver	r	The driver in use for streaming the data.
328	OnAncestorNotFound	rw	Handler called when the ancestor component cannot be found.
328	OnCreateComponent	rw	Handler called when a component needs to be created.
327	OnError	rw	Handler called when an error occurs.
329	OnFindComponentClass	rw	Handler called when a component class reference needs to be found.
327	OnFindMethod	rw	Handler to find or change a method address.
327	OnPropertyNotFound	rw	Handler for treating missing properties.
329	OnReadStringProperty	rw	Handler for translating strings when read from the stream.
328	OnReferenceName	rw	Handler called when another component is referenced.
328	OnSetMethodProperty	rw	Handler for setting method properties.
328	OnSetName	rw	Handler called when setting a component name.
326	Owner	rw	Owner of the component being read
327	Parent	rw	Parent of the component being read.

2.53.4 TReader.Create

Synopsis: Creates a new reader class

Declaration: `constructor Create(Stream: TStream;BufSize: Integer)`

Visibility: public

Description: Creates a new reader class

2.53.5 TReader.Destroy

Synopsis: Destroys a reader class.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys a reader class.

2.53.6 TReader.BeginReferences

Synopsis: Initializes the component referencing mechanism.

Declaration: `procedure BeginReferences`

Visibility: public

Description: Initializes the component referencing mechanism.

2.53.7 TReader.CheckValue

Synopsis: Raises an exception if the next value in the stream is not of type `Value`

Declaration:
```
procedure CheckValue(Value: TValueType)
```

Visibility: public

Description: Raises an exception if the next value in the stream is not of type `Value`

2.53.8 TReader.DefineProperty

Synopsis: Reads a user-defined property from the stream.

Declaration:
```
procedure DefineProperty(const Name: String;AReadData: TReaderProc;
                         WriteData: TWriterProc;HasData: Boolean)
                         ;  Override
```

Visibility: public

Description: Reads a user-defined property from the stream.

2.53.9 TReader.DefineBinaryProperty

Synopsis: Reads a user-defined binary property from the stream.

Declaration:
```
procedure DefineBinaryProperty(const Name: String;
                               AReadData: TStreamProc;
                               WriteData: TStreamProc;HasData: Boolean)
                               ;  Override
```

Visibility: public

Description: Reads a user-defined binary property from the stream.

2.53.10 TReader.EndOfList

Synopsis: Returns true if the stream contains an end-of-list marker.

Declaration:
```
function EndOfList : Boolean
```

Visibility: public

Description: Returns true if the stream contains an end-of-list marker.

2.53.11 TReader.EndReferences

Synopsis: Finalizes the component referencing mechanism.

Declaration:
```
procedure EndReferences
```

Visibility: public

Description: Finalizes the component referencing mechanism.

2.53.12 TReader.FixupReferences

Synopsis: Tries to resolve all unresolved component references.

Declaration: `procedure FixupReferences`

Visibility: public

Description: Tries to resolve all unresolved component references.

2.53.13 TReader.NextValue

Synopsis: Returns the type of the next value.

Declaration: `function NextValue : TValueType`

Visibility: public

Description: Returns the type of the next value.

2.53.14 TReader.Read

Synopsis: Read raw data from stream

Declaration: `procedure Read(var Buf;Count: LongInt);  Virtual`

Visibility: public

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TAbstractObjectReader.Read (227), TBinaryObjectReader.Read (246)

2.53.15 TReader.ReadBoolean

Synopsis: Reads a boolean from the stream.

Declaration: `function ReadBoolean : Boolean`

Visibility: public

Description: Reads a boolean from the stream.

2.53.16 TReader.ReadChar

Synopsis: Reads a character from the stream.

Declaration: `function ReadChar : Char`

Visibility: public

Description: Reads a character from the stream.

2.53.17 TReader.ReadWideChar

Synopsis: Read widechar from the stream

Declaration:
```
function ReadWideChar : WideChar
```

Visibility: public

Description: `TReader.ReadWideChar` reads a widechar from the stream. This actually reads a widestring and returns the first character.

See also: TReader.ReadWideString (326), TWriter.WriteWideChar (374)

2.53.18 TReader.ReadCollection

Synopsis: Reads a collection from the stream.

Declaration:
```
procedure ReadCollection(Collection: TCollection)
```

Visibility: public

Description: Reads a collection from the stream.

2.53.19 TReader.ReadComponent

Synopsis: Starts reading a component from the stream.

Declaration:
```
function ReadComponent(Component: TComponent) : TComponent
```

Visibility: public

Description: Starts reading a component from the stream.

2.53.20 TReader.ReadComponents

Synopsis: Starts reading child components from the stream.

Declaration:
```
procedure ReadComponents(AOwner: TComponent;AParent: TComponent;
                         Proc: TReadComponentsProc)
```

Visibility: public

Description: Starts reading child components from the stream.

2.53.21 TReader.ReadFloat

Synopsis: Reads a float from the stream.

Declaration:
```
function ReadFloat : Extended
```

Visibility: public

Description: Reads a float from the stream.

2.53.22 TReader.ReadSingle

Synopsis: Reads a single-type real from the stream.

Declaration: `function ReadSingle : Single`

Visibility: public

Description: Reads a single-type real from the stream.

2.53.23 TReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency`

Visibility: public

Description: `ReadCurrency` reads a currency typed value from the stream and returns the result. This method does nothing except call the driver method of the driver being used.

See also: TWriter.WriteCurrency (375)

2.53.24 TReader.ReadDate

Synopsis: Reads a date from the stream

Declaration: `function ReadDate : TDateTime`

Visibility: public

Description: Reads a date from the stream

2.53.25 TReader.ReadIdent

Synopsis: Reads an identifier from the stream.

Declaration: `function ReadIdent : String`

Visibility: public

Description: Reads an identifier from the stream.

2.53.26 TReader.ReadInteger

Synopsis: Reads an integer from the stream

Declaration: `function ReadInteger : LongInt`

Visibility: public

Description: Reads an integer from the stream

2.53.27 TReader.ReadInt64

Synopsis: Reads a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64`

Visibility: public

Description: Reads a 64-bit integer from the stream.

2.53.28 TReader.ReadListBegin

Synopsis: Checks for the beginning of a list.

Declaration: `procedure ReadListBegin`

Visibility: public

Description: Checks for the beginning of a list.

2.53.29 TReader.ReadListEnd

Synopsis: Checks for the end of a list.

Declaration: `procedure ReadListEnd`

Visibility: public

Description: Checks for the end of a list.

2.53.30 TReader.ReadRootComponent

Synopsis: Starts reading a root component.

Declaration: `function ReadRootComponent(ARoot: TComponent) : TComponent`

Visibility: public

Description: Starts reading a root component.

2.53.31 TReader.ReadString

Synopsis: Reads a string from the stream.

Declaration: `function ReadString : String`

Visibility: public

Description: Reads a string from the stream.

2.53.32 TReader.ReadWideString

Synopsis: Read a WideString value from the stream.

Declaration: `function ReadWideString : WideString`

Visibility: public

Description: `ReadWidestring` reads a widestring typed value from the stream and returns the result. This method does nothing except call the driver method of the driver being used.

See also: TWriter.WriteWideString (376)

2.53.33 TReader.ReadValue

Synopsis: Reads the next value type from the stream.

Declaration: `function ReadValue : TValueType`

Visibility: public

Description: Reads the next value type from the stream.

2.53.34 TReader.CopyValue

Synopsis: Copy a value to a writer.

Declaration: `procedure CopyValue(Writer: TWriter)`

Visibility: public

Description: Copy a value to a writer.

2.53.35 TReader.Driver

Synopsis: The driver in use for streaming the data.

Declaration: `Property Driver : TAbstractObjectReader`

Visibility: public

Access: Read

Description: The driver in use for streaming the data.

2.53.36 TReader.Owner

Synopsis: Owner of the component being read

Declaration: `Property Owner : TComponent`

Visibility: public

Access: Read,Write

Description: Owner of the component being read

2.53.37 TReader.Parent

Synopsis: Parent of the component being read.

Declaration: `Property Parent : TComponent`

Visibility: public

Access: Read,Write

Description: Parent of the component being read.

2.53.38 TReader.OnError

Synopsis: Handler called when an error occurs.

Declaration: `Property OnError : TReaderError`

Visibility: public

Access: Read,Write

Description: Handler called when an error occurs.

2.53.39 TReader.OnPropertyNotFound

Synopsis: Handler for treating missing properties.

Declaration: `Property OnPropertyNotFound : TPropertyNotFoundEvent`

Visibility: public

Access: Read,Write

Description: `OnPropertyNotFound` can be used to take appropriate action when a property is read from a stream and no such property is found in the RTTI information of the Instance that is being read from the stream. It can be set at runtime, or at designtime by an IDE.

For more information about the meaning of the various arguments to the event handler, see TPropertyNotFoundEvent (194).

See also: TPropertyNotFoundEvent (194), TReader.OnSetMethodProperty (328), TReader.OnReadStringProperty (329)

2.53.40 TReader.OnFindMethod

Synopsis: Handler to find or change a method address.

Declaration: `Property OnFindMethod : TFindMethodEvent`

Visibility: public

Access: Read,Write

Description: Handler to find or change a method address.

2.53.41 TReader.OnSetMethodProperty

Synopsis: Handler for setting method properties.

Declaration: `Property OnSetMethodProperty : TSetMethodPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnSetMethodProperty` can be set to handle the setting of method properties. This handler can be used by an IDE to prevent methods from actually being when an object is being streamed in the designer.

See also: TReader.OnReadStringProperty (329), TReader.OnPropertyNotFound (327)

2.53.42 TReader.OnSetName

Synopsis: Handler called when setting a component name.

Declaration: `Property OnSetName : TSetNameEvent`

Visibility: public

Access: Read,Write

Description: Handler called when setting a component name.

2.53.43 TReader.OnReferenceName

Synopsis: Handler called when another component is referenced.

Declaration: `Property OnReferenceName : TReferenceNameEvent`

Visibility: public

Access: Read,Write

Description: Handler called when another component is referenced.

2.53.44 TReader.OnAncestorNotFound

Synopsis: Handler called when the ancestor component cannot be found.

Declaration: `Property OnAncestorNotFound : TAncestorNotFoundEvent`

Visibility: public

Access: Read,Write

Description: Handler called when the ancestor component cannot be found.

2.53.45 TReader.OnCreateComponent

Synopsis: Handler called when a component needs to be created.

Declaration: `Property OnCreateComponent : TCreateComponentEvent`

Visibility: public

Access: Read,Write

Description: Handler called when a component needs to be created.

2.53.46 TReader.OnFindComponentClass

Synopsis: Handler called when a component class reference needs to be found.

Declaration: `Property OnFindComponentClass : TFindComponentClassEvent`

Visibility: public

Access: Read,Write

Description: Handler called when a component class reference needs to be found.

2.53.47 TReader.OnReadStringProperty

Synopsis: Handler for translating strings when read from the stream.

Declaration: `Property OnReadStringProperty : TReadWriteStringPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnReadStringProperty` is called whenever a string property is read from the stream. It can be used e.g. by a translation mechanism to translate the strings on the fly, when a form is loaded. See TReadWriteStringPropertyEvent (195) for a description of the various parameters.

See also: TReader.OnPropertyNotFound (327), TReader.OnSetMethodProperty (328), TReadWriteStringPropertyEvent (195)

2.54 TRecall

2.54.1 Description

`TRecall` is a helper class used to copy published properties of a class (the reference object) in another class (the storage object). The reference object and storage object must be assignable to each other.

The `TRecall` can be used to store the state of a persistent class, and restore it at a later time.

When a `TRecall` object is created, it gets passed a reference instance and a storage instance. It immediatly stores the properties of the reference object in the storage object.

The Store (330) method can be called throughout the lifetime of the reference object to update the stored properties.

When the `TRecall` instance is destroyed then the properties are copied from the storage object to the reference object. The storage object is freed automatically.

If the properties should not be copied back from the storage to the reference object, the Forget (330) can be called.

2.54.2 Method overview

Page	Property	Description
330	Create	Creates a new instance of `TRecall`.
330	Destroy	Copies the stored properties to the reference object and destroys the `TRecall` instance.
330	Forget	Clear the reference property.
330	Store	Assigns the reference instance to the storage instance.

2.54.3 Property overview

Page	Property	Access	Description
331	Reference	r	The reference object.

2.54.4 TRecall.Create

Synopsis: Creates a new instance of `TRecall`.

Declaration: `constructor Create(AStorage: TPersistent;AReference: TPersistent)`

Visibility: public

Description: `Create` creates a new instance of `TRecall` and initializes the Reference and Storage instances. It calls Store (330) to assign the reference object properties to the storage instance.

See also: TRecall.Store (330), TRecall.Destroy (330)

2.54.5 TRecall.Destroy

Synopsis: Copies the stored properties to the reference object and destroys the `TRecall` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` assigns the storage instance to the reference instance, if the latter is still valid. After this, it frees the storage and calls the inherited destroy.

Errors: `Destroy` does not check whether the reference (331) instance is still valid. If the reference pointer was invalidated, call TRecall.Forget (330) to clear the reference instance.

See also: TRecall.Store (330), TRecall.Forget (330)

2.54.6 TRecall.Store

Synopsis: Assigns the reference instance to the storage instance.

Declaration: `procedure Store`

Visibility: public

Description: `Store` assigns the reference instance to the storage instance. This will only work if the two classes can be assigned to each other.

This method can be used to refresh the storage.

Errors: `Store` does not check whether the reference (331) instance is still valid. If the reference pointer was invalidated, call TRecall.Forget (330) to clear the reference instance.

2.54.7 TRecall.Forget

Synopsis: Clear the reference property.

Declaration: `procedure Forget`

Visibility: public

Description: `Forget` sets the Reference (185) property to `Nil`. When the `TRecall` instance is destroyed, the reference instance will not be restored.

Note that after a call to `Forget`, a call to Store (330) has no effect.

Errors: None.

See also: TRecall.Reference (331), TRecall.Store (330), TRecall.Destroy (330)

2.54.8 TRecall.Reference

Synopsis: The reference object.

Declaration: `Property Reference : TPersistent`

Visibility: public

Access: Read

Description: `Reference` is the instance of the reference object. Do not free the reference directly. Call Forget (330) to clear the reference and then free the reference object.

See also: TRecall.Forget (330)

2.55 TResourceStream

2.55.1 Description

Stream that reads its data from a resource object.

2.55.2 Method overview

Page	Property	Description
331	Create	Creates a new instance of a resource stream.
331	CreateFromID	Creates a new instance of a resource stream with resource
332	Destroy	Destroys the instance of the resource stream.

2.55.3 TResourceStream.Create

Synopsis: Creates a new instance of a resource stream.

Declaration:
```
constructor Create(Instance: THandle;const ResName: String;
                   ResType: PChar)
```

Visibility: public

Description: Creates a new instance of a resource stream.

2.55.4 TResourceStream.CreateFromID

Synopsis: Creates a new instance of a resource stream with resource

Declaration:
```
constructor CreateFromID(Instance: THandle;ResID: Integer;
                         ResType: PChar)
```

Visibility: public

Description: Creates a new instance of a resource stream with resource

2.55.5 TResourceStream.Destroy

Synopsis: Destroys the instance of the resource stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the instance of the resource stream.

2.56 TStream

2.56.1 Description

`TStream` is the base class for all streaming classes. It defines methods for reading (333), writing (333) from and to streams, as well as functions to determine the size of the stream as well as the current position of the stream.

Descendent classes such as TMemoryStream (306) or TFileStream (284) then override these methods to write streams to memory or file.

2.56.2 Method overview

Page	Property	Description
335	CopyFrom	Copy data from one stream to another
337	FixupResourceHeader	Not implemented in FPC
333	Read	Reads data from the stream to a buffer and returns the number of bytes read.
338	ReadAnsiString	Read an ansistring from the stream and return its value.
334	ReadBuffer	Reads data from the stream to a buffer
337	ReadByte	Read a byte from the stream and return its value.
335	ReadComponent	Reads component data from a stream
335	ReadComponentRes	Reads component data and resource header from a stream
338	ReadDWord	Read a DWord from the stream and return its value.
337	ReadResHeader	Read a resource header from the stream.
338	ReadWord	Read a word from the stream and return its value.
333	Seek	Sets the current position in the stream
333	Write	Writes data from a buffer to the stream and returns the number of bytes written.
339	WriteAnsiString	Write an ansistring to the stream.
334	WriteBuffer	Writes data from the stream to the buffer
339	WriteByte	Write a byte to the stream.
336	WriteComponent	Write component data to the stream
336	WriteComponentRes	Write resource header and component data to a stream
336	WriteDescendent	Write component data to a stream, relative to an ancestor
336	WriteDescendentRes	Write resource header and component data to a stream, relative to an ancestor
339	WriteDWord	Write a DWord to the stream.
337	WriteResourceHeader	Write resource header to the stream
339	WriteWord	Write a word to the stream.

2.56.3 Property overview

Page	Property	Access	Description
340	Position	rw	The current position in the stream.
340	Size	rw	The current size of the stream.

2.56.4 TStream.Read

Synopsis: Reads data from the stream to a buffer and returns the number of bytes read.

Declaration:
```
function Read(var Buffer;Count: LongInt) : LongInt;  Virtual
```

Visibility: public

Description: `Read` attempts to read `Count` from the stream to `Buffer` and returns the number of bytes actually read.

This method should be used when the number of bytes is not determined. If a specific number of bytes is expected, use TSTream.ReadBuffer (334) instead.

As implemented in `TStream`, `Read` does nothing but raises an EStreamError (216) exception to indicate that reading is not supported. Descendent classes that allow reading must override this method to do the actual reading.

Errors: In case a descendent class does not allow reading from the stream, an exception is raised.

See also: TStream.Write (333), TStream.ReadBuffer (334)

2.56.5 TStream.Write

Synopsis: Writes data from a buffer to the stream and returns the number of bytes written.

Declaration:
```
function Write(const Buffer;Count: LongInt) : LongInt;  Virtual
```

Visibility: public

Description: `Write` attempts to write `Count` bytes from `Buffer` to the stream. It returns the actual number of bytes written to the stream.

This method should be used when the number of bytes that should be written is not determined. If a specific number of bytes should be written, use TSTream.WriteBuffer (334) instead.

As implemented in `TStream`, `Write` does nothing but raises EStreamError (216) exception to indicate that writing is not supported. Descendent classes that allow writing must override this method to do the actual writing.

Errors: In case a descendent class does not allow writing to the stream, an exception is raised.

See also: TStream.Read (333), TStream.WriteBuffer (334)

2.56.6 TStream.Seek

Synopsis: Sets the current position in the stream

Declaration:
```
function Seek(Offset: LongInt;Origin: Word) : LongInt;  Virtual
                ;  Overload
function Seek(const Offset: Int64;Origin: TSeekOrigin) : Int64;  Virtual
                ;  Overload
```

Visibility: public

Description: `Seek` sets the position of the stream to `Offset` bytes from `Origin`. `Origin` can have one of the following values:

Table 2.21:

Constant	Meaning
soFromBeginning	Set the position relative to the start of the stream.
soFromCurrent	Set the position relative to the beginning of the stream.
soFromEnd	Set the position relative to the end of the stream.

`Offset` should be negative when the origin is `SoFromEnd`. It should be positive for `soFromBeginning` and can have both signs for `soFromCurrent`

This is an abstract method, which must be overridden by descendent classes. They may choose not to implement this method for all values of `Origin` and `Offset`.

Errors: An exception may be raised if this method is called with an invalid pair of `Offset`,`Origin` values. e.g. a negative `offset` for `soFromBeginning`.

See also: TStream.Position (340)

2.56.7 TStream.ReadBuffer

Synopsis: Reads data from the stream to a buffer

Declaration: `procedure ReadBuffer(var Buffer;Count: LongInt)`

Visibility: public

Description: `ReadBuffer` reads `Count` bytes of the stream into `Buffer`. If the stream does not contain `Count` bytes, then an exception is raised.

`ReadBuffer` should be used to read in a fixed number of bytes, such as when reading structures or the content of variables. If the number of bytes is not determined, use TStream.Read (333) instead. `ReadBuffer` uses `Read` internally to do the actual reading.

Errors: If the stream does not allow to read `Count` bytes, then an exception is raised.

See also: TStream.Read (333), TStream.WriteBuffer (334)

2.56.8 TStream.WriteBuffer

Synopsis: Writes data from the stream to the buffer

Declaration: `procedure WriteBuffer(const Buffer;Count: LongInt)`

Visibility: public

Description: `WriteBuffer` writes `Count` bytes to the stream from `Buffer`. If the stream does not allow `Count` bytes to be written, then an exception is raised.

`WriteBuffer` should be used to read in a fixed number of bytes, such as when writing structures or the content of variables. If the number of bytes is not determined, use TStream.Write (333) instead. `WriteBuffer` uses `Write` internally to do the actual reading.

Errors: If the stream does not allow to write `Count` bytes, then an exception is raised.

See also: TStream.Write (333), TStream.ReadBuffer (334)

2.56.9 TStream.CopyFrom

Synopsis: Copy data from one stream to another

Declaration: `function CopyFrom(Source: TStream;Count: Int64) : Int64`

Visibility: public

Description: `CopyFrom` reads `Count` bytes from `Source` and writes them to the current stream. This updates the current position in the stream. After the action is completed, the number of bytes copied is returned.

This can be used to quickly copy data from one stream to another or to copy the whole contents of the stream.

See also: TStream.Read (333), TStream.Write (333)

2.56.10 TStream.ReadComponent

Synopsis: Reads component data from a stream

Declaration: `function ReadComponent(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponent` reads a component state from the stream and transfers this state to `Instance`. If `Instance` is nil, then it is created first based on the type stored in the stream. `ReadComponent` returns the component as it is read from the stream.

`ReadComponent` simply creates a TReader (318) object and calls its ReadRootComponent (325) method.

Errors: If an error occurs during the reading of the component, an EFilerError (215) exception is raised.

See also: TStream.WriteComponent (336), TStream.ReadComponentRes (335), TReader.ReadRootComponent (325)

2.56.11 TStream.ReadComponentRes

Synopsis: Reads component data and resource header from a stream

Declaration: `function ReadComponentRes(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponentRes` reads a resource header from the stream, and then calls ReadComponent (335) to read the component state from the stream into `Instance`.

This method is usually called by the global streaming method when instantiating forms and datamodules as created by an IDE. It should be used mainly on Windows, to store components in Windows resources.

Errors: If an error occurs during the reading of the component, an EFilerError (215) exception is raised.

See also: TStream.ReadComponent (335), TStream.WriteComponentRes (336)

2.56.12 TStream.WriteComponent

Synopsis: Write component data to the stream

Declaration: `procedure WriteComponent(Instance: TComponent)`

Visibility: public

Description: `WriteComponent` writes the published properties of `Instance` to the stream, so they can later be read with TStream.ReadComponent (335). This method is intended to be used by an IDE, to preserve the state of a form or datamodule as designed in the IDE.

`WriteComponent` simply calls WriteDescendent (336) with `Nil` ancestor.

See also: TStream.ReadComponent (335), TStream.WriteComponentRes (336)

2.56.13 TStream.WriteComponentRes

Synopsis: Write resource header and component data to a stream

Declaration: `procedure WriteComponentRes(const ResName: String;Instance: TComponent)`

Visibility: public

Description: `WriteComponentRes` writes a `ResName` resource header to the stream and then calls WriteComponent (336) to write the published properties of `Instance` to the stream.

This method is intened for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

See also: TStream.WriteComponent (336), TStream.ReadComponentRes (335)

2.56.14 TStream.WriteDescendent

Synopsis: Write component data to a stream, relative to an ancestor

Declaration: `procedure WriteDescendent(Instance: TComponent;Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendent` writes the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

`WriteDescendent` creates a TWriter (372) object and calls its WriteDescendent (374) object. The writer is passed a binary driver object (249) by default.

2.56.15 TStream.WriteDescendentRes

Synopsis: Write resource header and component data to a stream, relative to an ancestor

Declaration:
```
procedure WriteDescendentRes(const ResName: String;Instance: TComponent;
                             Ancestor: TComponent)
```

Visibility: public

Description: `WriteDescendentRes` writes a `ResName` resource header, and then calls WriteDescendent (336) to write the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

This method is intened for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

2.56.16 TStream.WriteResourceHeader

Synopsis: Write resource header to the stream

Declaration:
```
procedure WriteResourceHeader(const ResName: String;
                              var FixupInfo: Integer)
```

Visibility: public

Description: `WriteResourceHeader` writes a resource-file header for a resource called `ResName`. It returns in `FixupInfo` the argument that should be passed on to TStream.FixupResourceHeader (337).

`WriteResourceHeader` should not be used directly. It is called by the TStream.WriteComponentRes (336) and TStream.WriteDescendentRes (336) methods.

See also: TStream.FixupResourceHeader (337), TStream.WriteComponentRes (336), TStream.WriteDescendentRes (336)

2.56.17 TStream.FixupResourceHeader

Synopsis: Not implemented in FPC

Declaration:
```
procedure FixupResourceHeader(FixupInfo: Integer)
```

Visibility: public

Description: `FixupResourceHeader` is used to write the size of the resource after a component was written to stream. The size is determined from the current position, and it is written at position `FixupInfo`. After that the current position is restored.

`FixupResourceHeader` should never be called directly; it is handled by the streaming system.

See also: TStream.WriteResourceHeader (337), TStream.WriteComponentRes (336), TStream.WriteDescendentRes (336)

2.56.18 TStream.ReadResHeader

Synopsis: Read a resource header from the stream.

Declaration:
```
procedure ReadResHeader
```

Visibility: public

Description: `ReadResourceHeader` reads a reasource file header from the stream. It positions the stream just beyond the header.

`ReadResourceHeader` should not be called directly, it is called by the streaming system when needed.

Errors: If the resource header is invalid an EInvalidImage (215) exception is raised.

See also: TStream.ReadComponentRes (335), EInvalidImage (215)

2.56.19 TStream.ReadByte

Synopsis: Read a byte from the stream and return its value.

Declaration:
```
function ReadByte : Byte
```

Visibility: public

Description: `ReadByte` reads one byte from the stream and returns its value.

Errors: If the byte cannot be read, a EStreamError (216) exception will be raised. This is a utility function which symply calls the Read (333) function.

See also: TStream.Read (333), TStream.WriteByte (339), TStream.ReadWord (338), TStream.ReadDWord (338), TStream.ReadAnsiString (338)

2.56.20 TStream.ReadWord

Synopsis: Read a word from the stream and return its value.

Declaration: `function ReadWord : Word`

Visibility: public

Description: `ReadWord` reads one Word (i.e. 2 bytes) from the stream and returns its value. This is a utility function which symply calls the Read (333) function.

Errors: If the word cannot be read, a EStreamError (216) exception will be raised.

See also: TStream.Read (333), TStream.WriteWord (339), TStream.ReadByte (337), TStream.ReadDWord (338), TStream.ReadAnsiString (338)

2.56.21 TStream.ReadDWord

Synopsis: Read a DWord from the stream and return its value.

Declaration: `function ReadDWord : Cardinal`

Visibility: public

Description: `ReadDWord` reads one DWord (i.e. 4 bytes) from the stream and returns its value. This is a utility function which simply calls the Read (333) function.

Errors: If the DWord cannot be read, a EStreamError (216) exception will be raised.

See also: TStream.Read (333), TStream.WriteDWord (339), TStream.ReadByte (337), TStream.ReadWord (338), TStream.ReadAnsiString (338)

2.56.22 TStream.ReadAnsiString

Synopsis: Read an ansistring from the stream and return its value.

Declaration: `function ReadAnsiString : String`

Visibility: public

Description: `ReadAnsiString` reads an ansistring from the stream and returns its value. This is a utility function which simply calls the read function several times. The Ansistring should be stored as 4 bytes (a DWord) representing the length of the string, and then the string value itself. The WriteAnsiString (339) function writes an ansistring in such a format.

Errors: If the AnsiString cannot be read, a EStreamError (216) exception will be raised.

See also: TStream.Read (333), TStream.WriteAnsiString (339), TStream.ReadByte (337), TStream.ReadWord (338), TStream.ReadDWord (338)

2.56.23 TStream.WriteByte

Synopsis: Write a byte to the stream.

Declaration: `procedure WriteByte(b: Byte)`

Visibility: public

Description: `WriteByte` writes the byte `B` to the stream. This is a utility function which simply calls the Write (333) function. The byte can be read from the stream using the ReadByte (337) function.

Errors: If an error occurs when attempting to write, an EStreamError (216) exception will be raised.

See also: TStream.Write (333), TStream.ReadByte (337), TStream.WriteWord (339), TStream.WriteDWord (339), TStream.WriteAnsiString (339)

2.56.24 TStream.WriteWord

Synopsis: Write a word to the stream.

Declaration: `procedure WriteWord(w: Word)`

Visibility: public

Description: `WriteWord` writes the word `W` (i.e. 2 bytes) to the stream. This is a utility function which simply calls the Write (333) function. The word can be read from the stream using the ReadWord (338) function.

Errors: If an error occurs when attempting to write, an EStreamError (216) exception will be raised.

See also: TStream.Write (333), TStream.ReadWord (338), TStream.WriteByte (339), TStream.WriteDWord (339), TStream.WriteAnsiString (339)

2.56.25 TStream.WriteDWord

Synopsis: Write a DWord to the stream.

Declaration: `procedure WriteDWord(d: Cardinal)`

Visibility: public

Description: `WriteDWord` writes the DWord `D` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the Write (333) function. The DWord can be read from the stream using the ReadDWord (338) function.

Errors: If an error occurs when attempting to write, an EStreamError (216) exception will be raised.

See also: TStream.Write (333), TStream.ReadDWord (338), TStream.WriteByte (339), TStream.WriteWord (339), TStream.WriteAnsiString (339)

2.56.26 TStream.WriteAnsiString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(const S: String)`

Visibility: public

Description: `WriteAnsiString` writes the AnsiString `S` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the Write (333) function. The ansistring is written as a 4 byte length specifier, followed by the ansistring's content. The ansistring can be read from the stream using the ReadAnsiString (338) function.

Errors: If an error occurs when attempting to write, an EStreamError (216) exception will be raised.

See also: TStream.Write (333), TStream.ReadAnsiString (338), TStream.WriteByte (339), TStream.WriteWord (339), TStream.WriteDWord (339)

2.56.27 TStream.Position

Synopsis: The current position in the stream.

Declaration: `Property Position : Int64`

Visibility: public

Access: Read,Write

Description: `Position` can be read to determine the current position in the stream. It can be written to to set the (absolute) position in the stream. The position is zero-based, so to set the position at the beginning of the stream, the position must be set to zero.

Remark: Not all `TStream` descendants support setting the position in the stream, so this should be used with care.

Errors: Some descendents may raise an EStreamError (216) exception if they do not support setting the stream position.

See also: TStream.Size (340), TStream.Seek (333)

2.56.28 TStream.Size

Synopsis: The current size of the stream.

Declaration: `Property Size : Int64`

Visibility: public

Access: Read,Write

Description: `Size` can be read to determine the stream size or to set the stream size.

Remark: Not all descendents of `TStream` support getting or setting the stream size; they may raise an exception if the `Size` property is read or set.

See also: TStream.Position (340), TStream.Seek (333)

2.57 TStreamAdapter

2.57.1 Description

Implements IStream for TStream (332) descendents

2.57.2 Method overview

Page	Property	Description
344	Clone	Clone the stream
343	Commit	Commit data to the stream
343	CopyTo	Copy data to destination stream
341	Create	Create a new instance of `TStreamAdapter`
341	Destroy	Free the `TStreamAdapter` instance
343	LockRegion	Lock a region of the stream
342	Read	Read from the stream.
343	Revert	Revert operations on the stream
342	Seek	Set the stream position
342	SetSize	Set the stream size
344	Stat	Return statistical data from the stream
344	UnlockRegion	Unlock a region of the stream
342	Write	Write to the stream

2.57.3 Property overview

Page	Property	Access	Description
344	Stream	r	Stream on which adaptor works
345	StreamOwnership	rw	Determines what happens with the stream when the adaptor is freed

2.57.4 TStreamAdapter.Create

Synopsis: Create a new instance of `TStreamAdapter`

Declaration: `constructor Create(Stream: TStream;Ownership: TStreamOwnership)`

Visibility: public

Description: `Create` creates a new instance of `TStreamAdaptor`. It initializes TStreamAdapter.Stream (344) with `Stream` and initializes StreamOwnerShip (345) with `Ownership`.

`TStreamAdapter` is an abstract class: descendents must be created that implement the actual functionality.

Errors:

See also: TStreamAdapter.StreamOwnerShip (345), TStreamAdapter.Stream (344)

2.57.5 TStreamAdapter.Destroy

Synopsis: Free the `TStreamAdapter` instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Explicitly free the `TStreamAdapter` instance. Normally, this is done automatically if a reference to the `IStream` interface is freed.

2.57.6 TStreamAdapter.Read

Synopsis: Read from the stream.

Declaration:
```
function Read(pv: Pointer;cb: DWORD;pcbRead: PDWORD) : HResult;  Virtual
```

Visibility: public

Description: `Read` implements IStream.Read (185) by reading from the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Read (185)

2.57.7 TStreamAdapter.Write

Synopsis: Write to the stream

Declaration:
```
function Write(pv: Pointer;cb: DWORD;pcbWritten: PDWORD) : HResult
                    ;  Virtual
```

Visibility: public

Description: `Write` implements IStream.Write (185) by writing to the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Write (185)

2.57.8 TStreamAdapter.Seek

Synopsis: Set the stream position

Declaration:
```
function Seek(dlibMove: Largeint;dwOrigin: LongInt;
                  out libNewPosition: Largeint) : HResult;  Virtual
```

Visibility: public

Description: `Seek` implements IStream.Seek (185) by setting the position of the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Seek (185)

2.57.9 TStreamAdapter.SetSize

Synopsis: Set the stream size

Declaration:
```
function SetSize(libNewSize: Largeint) : HResult;  Virtual
```

Visibility: public

Description: `SetSize` implements IStream.Setsize (185) by setting the size of the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Setsize (185)

2.57.10 TStreamAdapter.CopyTo

Synopsis: Copy data to destination stream

Declaration:
```
function CopyTo(stm: IStream;cb: Largeint;out cbRead: Largeint;
                out cbWritten: Largeint) : HResult;  Virtual
```

Visibility: public

Description: `CopyTo` implements IStream.CopyTo (185).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

2.57.11 TStreamAdapter.Commit

Synopsis: Commit data to the stream

Declaration:
```
function Commit(grfCommitFlags: LongInt) : HResult;  Virtual
```

Visibility: public

Description: `Commit` implements IStream.Commit (185).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Commit (185)

2.57.12 TStreamAdapter.Revert

Synopsis: Revert operations on the stream

Declaration:
```
function Revert : HResult;  Virtual
```

Visibility: public

Description: `Revert` implements IStream.Revert (185).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Revert (185)

2.57.13 TStreamAdapter.LockRegion

Synopsis: Lock a region of the stream

Declaration:
```
function LockRegion(libOffset: Largeint;cb: Largeint;
                    dwLockType: LongInt) : HResult;  Virtual
```

Visibility: public

Description: `LockRegion` implements IStream.LockRegion (185).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.LockRegion (185)

2.57.14 TStreamAdapter.UnlockRegion

Synopsis: Unlock a region of the stream

Declaration:
```
function UnlockRegion(libOffset: Largeint;cb: Largeint;
                      dwLockType: LongInt) : HResult;  Virtual
```

Visibility: public

Description: `UnLockRegion` implements IStream.UnLockRegion (185).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.UnLockRegion (185)

2.57.15 TStreamAdapter.Stat

Synopsis: Return statistical data from the stream

Declaration:
```
function Stat(out statstg: TStatStg;grfStatFlag: LongInt) : HResult
                ;  Virtual
```

Visibility: public

Description: `Stat` implements IStream.Stat (185).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Stat (185)

2.57.16 TStreamAdapter.Clone

Synopsis: Clone the stream

Declaration:
```
function Clone(out stm: IStream) : HResult;  Virtual
```

Visibility: public

Description: `Clone` implements IStream.Clone (185).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: IStream.Clone (185)

2.57.17 TStreamAdapter.Stream

Synopsis: Stream on which adaptor works

Declaration:
```
Property Stream : TStream
```

Visibility: public

Access: Read

Description: This is the stream on which the adaptor works. It was specified at rreation.

2.57.18 TStreamAdapter.StreamOwnership

Synopsis: Determines what happens with the stream when the adaptor is freed

Declaration: `Property StreamOwnership : TStreamOwnership`

Visibility: public

Access: Read,Write

Description: `StreamOwnership` determines what happens when the adaptor

2.58 TStringList

2.58.1 Description

`TStringList` is a descendent class of TStrings (349) that implements all of the abstract methods introduced there. It also introduces some additional methods:

- Sort the list, or keep the list sorted at all times
- Special handling of duplicates in sorted lists
- Notification of changes in the list

2.58.2 Method overview

Page	Property	Description
346	Add	Implements the TStrings.Add (351) function.
346	Clear	Implements the TStrings.Clear (353) function.
348	CustomSort	
346	Delete	Implements the TStrings.Delete (353) function.
345	Destroy	Destroys the stringlist.
346	Exchange	Implements the TStrings.Exchange (354) function.
347	Find	Locates the index for a given string in sorted lists.
347	IndexOf	Overrides the TStrings.IndexOf (355) property.
347	Insert	Overrides the TStrings.Insert (356) method.
347	Sort	Sorts the strings in the list.

2.58.3 Property overview

Page	Property	Access	Description
349	CaseSensitive	rw	
348	Duplicates	rw	Describes the behaviour of a sorted list with respect to duplicate strings.
349	OnChange	rw	Event triggered after the list was modified.
349	OnChanging	rw	Event triggered when the list is about to be modified.
348	Sorted	rw	Determines whether the list is sorted or not.

2.58.4 TStringList.Destroy

Synopsis: Destroys the stringlist.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the stringlist, release all memory allocated for the storage of the strings, and then calls the inherited destroy method.

Remark: Any objects associated to strings in the list will *not* be destroyed; it is the responsability of the caller to destroy all objects associated with strings in the list.

2.58.5 TStringList.Add

Synopsis: Implements the TStrings.Add (351) function.

Declaration: `function Add(const S: String) : Integer;  Override`

Visibility: public

Description: `Add` will add `S` to the list. If the list is sorted and the string `S` is already present in the list and TStringList.Duplicates (348) is `dupError` then an EStringListError (217) exception is raised. If `Duplicates` is set to `dupIgnore` then the return value is underfined.

If the list is sorted, new strings will not necessarily be added to the end of the list, rather they will be inserted at their alphabetical position.

Errors: If the list is sorted and the string `S` is already present in the list and TStringList.Duplicates (348) is `dupError` then an EStringListError (217) exception is raised.

See also: TStringList.Insert (347), TStringList.Duplicates (348)

2.58.6 TStringList.Clear

Synopsis: Implements the TStrings.Clear (353) function.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: Implements the TStrings.Clear (353) function.

2.58.7 TStringList.Delete

Synopsis: Implements the TStrings.Delete (353) function.

Declaration: `procedure Delete(Index: Integer);  Override`

Visibility: public

Description: Implements the TStrings.Delete (353) function.

2.58.8 TStringList.Exchange

Synopsis: Implements the TStrings.Exchange (354) function.

Declaration: `procedure Exchange(Index1: Integer;Index2: Integer);  Override`

Visibility: public

Description: `Exchange` will exchange two items in the list as described in TStrings.Exchange (354).

Remark: `Exchange` will not check whether the list os sorted or not; if `Exchange` is called on a sorted list and the strings are not identical, the sort order of the list will be destroyed.

See also: TStringList.Sorted (348), TStrings.Exchange (354)

2.58.9 TStringList.Find

Synopsis: Locates the index for a given string in sorted lists.

Declaration: `function Find(const S: String;var Index: Integer) : Boolean;  Virtual`

Visibility: public

Description: `Find` returns `True` if the string `S` is present in the list. Upon exit, the `Index` parameter will contain the position of the string in the list. If the string is not found, the function will return `False` and `Index` will contain the position where the string will be inserted if it is added to the list.

Remark:

1.Use this method only on sorted lists. For unsorted lists, use TStringList.IndexOf (347) instead.

2.Find uses a binary search method to locate the string

2.58.10 TStringList.IndexOf

Synopsis: Overrides the TStrings.IndexOf (355) property.

Declaration: `function IndexOf(const S: String) : Integer;  Override`

Visibility: public

Description: `IndexOf` overrides the ancestor method TStrings.indexOf (355). It tries to optimize the search by executing a binary search if the list is sorted. The function returns the position of `S` if it is found in the list, or -1 if the string is not found in the list.

See also: TStrings.IndexOf (355), TStringList.Find (347)

2.58.11 TStringList.Insert

Synopsis: Overrides the TStrings.Insert (356) method.

Declaration: `procedure Insert(Index: Integer;const S: String);  Override`

Visibility: public

Description: `Insert` will insert the string `S` at position `Index` in the list. If the list is sorted, an EStringListError (217) exception will be raised instead. `Index` is a zero-based position.

Errors: If `Index` contains an invalid value (less than zero or larger than `Count`, or the list is sorted, an EStringListError (217) exception will be raised.

See also: TStringList.Add (346), TStrings.Insert (356), TStringList.InsertObject (345)

2.58.12 TStringList.Sort

Synopsis: Sorts the strings in the list.

Declaration: `procedure Sort;  Virtual`

Visibility: public

Description: `Sort` will sort the strings in the list using the quicksort algorithm. If the list has its TStringList.Sorted (348) property set to `True` then nothing will be done.

See also: TStringList.Sorted (348)

2.58.13 TStringList.CustomSort

Synopsis:

Declaration: `procedure CustomSort(CompareFn: TStringListSortCompare); Virtual`

Visibility: public

Description:

2.58.14 TStringList.Duplicates

Synopsis: Describes the behaviour of a sorted list with respect to duplicate strings.

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` describes what to do in case a duplicate value is added to the list:

Table 2.22:

dupIgnore	Duplicate values will not be be added to the list, but no error will be triggered.
dupError	If an attempt is made to add a duplicate value to the list, an EStringListError (217) exception is raised.
dupAccept	Duplicate values can be added to the list.

If the stringlist is not sorted, the `Duplicates` setting is ignored.

2.58.15 TStringList.Sorted

Synopsis: Determines whether the list is sorted or not.

Declaration: `Property Sorted : Boolean`

Visibility: public

Access: Read,Write

Description: `Sorted` can be set to `True` in order to cause the list of strings to be sorted. Further additions to the list will be inserted at the correct position so the list remains sorted at all times. Setting the property to `False` has no immediate effect, but will allow strings to be inserted at any position.

Remark:

1. When `Sorted` is `True`, TStringList.Insert (347) cannot be used. For sorted lists, TStringList.Add (346) should be used instead.
2. If `Sorted` is `True`, the TStringList.Duplicates (348) setting has effect. This setting is ignored when `Sorted` is `False`.

See also: TStringList.Sort (347), TStringList.Duplicates (348), TStringList.Add (346), TstringList.Insert (347)

2.58.16 TStringList.CaseSensitive

Synopsis:

Declaration: `Property CaseSensitive : Boolean`

Visibility: public

Access: Read,Write

Description: Indicates whether locating strings happens in a case sensitive manner.

2.58.17 TStringList.OnChange

Synopsis: Event triggered after the list was modified.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChange` can be assigned to respond to changes that have occurred in the list. The handler is called whenever strings are added, moved, modified or deleted from the list.

The `Onchange` event is triggered after the modification took place. When the modification is about to happen, an TstringList.OnChanging (349) event occurs.

See also: TStringList.OnChanging (349)

2.58.18 TStringList.OnChanging

Synopsis: Event triggered when the list is about to be modified.

Declaration: `Property OnChanging : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChanging` can be assigned to respond to changes that will occurred in the list. The handler is called whenever strings will be added, moved, modified or deleted from the list.

The `Onchanging` event is triggered before the modification will take place. When the modification has happened, an TstringList.OnChange (349) event occurs.

See also: TStringList.OnChange (349)

2.59 TStrings

2.59.1 Description

`TStrings` implements an abstract class to manage an array of strings. It introduces methods to set and retrieve strings in the array, searching for a particular string, concatenating the strings and so on. It also allows an arbitrary object to be associated with each string.

It also introduces methods to manage a series of `name=value` settings, as found in many configuration files.

An instance of `TStrings` is never created directly, instead a descendent class such as TStringList (345) should be created. This is because `TStrings` is an abstract class which does not implement all methods; `TStrings` also doesn't store any strings, this is the functionality introduced in descendents such as TStringList (345).

2.59.2 Method overview

Page	Property	Description
351	Add	Add a string to the list
351	AddObject	Add a string and associated object to the list.
352	AddStrings	Add contents of another stringlist to this list.
352	Append	Add a string to the list.
352	Assign	Assign the contents of another stringlist to this one.
352	BeginUpdate	Mark the beginning of an update batch.
353	Clear	Removes all strings and associated objects from the list.
353	Delete	Delete a string from the list.
351	Destroy	Frees all strings and objects, and removes the list from memory.
354	EndUpdate	Mark the end of an update batch.
354	Equals	Compares the contents of two stringlists.
354	Exchange	Exchanges two strings in the list.
359	ExtractName	Extract the name part of a string
358	GetNameValue	Return both name and value of a name,value pair based on it's index.
355	GetText	Returns the contents as a PChar
355	IndexOf	Find a string in the list and return its position.
355	IndexOfName	Finds the index of a name in the name-value pairs.
355	IndexOfObject	Finds an object in the list and returns its index.
356	Insert	Insert a string in the list.
356	InsertObject	Insert a string and associated object in the list.
356	LoadFromFile	Load the contents of a file as a series of strings.
357	LoadFromStream	Load the contents of a stream as a series of strings.
357	Move	Move a string from one place in the list to another.
358	SaveToFile	Save the contents of the list to a file.
358	SaveToStream	Save the contents of the string to a stream.
358	SetText	Set the contents of the list from a PChar.

2.59.3 Property overview

Page	Property	Access	Description
360	Capacity	rw	Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.
361	CommaText	rw	Contents of the list as a comma-separated string.
361	Count	r	Number of strings in the list.
359	DelimitedText	rw	Get or set all strings in the list in a delimited form.
359	Delimiter	rw	Delimiter character used in DelimitedText (359).
362	Names	r	Name parts of the name-value pairs in the list.
360	NameValueSeparator	rw	Value of the character used to separate name,value pairs
362	Objects	rw	Indexed access to the objects associated with the strings in the list.
360	QuoteChar	rw	Quote character used in DelimitedText (359).
363	Strings	rw	Indexed access to the strings in the list.
364	StringsAdapter	rw	Not implemented in Free Pascal.
363	Text	rw	Contents of the list as one big string.
359	TextLineBreakStyle	rw	Determines which line breaks to use in the Text (363) property
360	ValueFromIndex	rw	
362	Values	rw	Value parts of the name-value pairs in the list.

2.59.4 TStrings.Destroy

Synopsis: Frees all strings and objects, and removes the list from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` is the destructor of `TStrings` it does nothing except calling the inherited destructor.

2.59.5 TStrings.Add

Synopsis: Add a string to the list

Declaration: `function Add(const S: String) : Integer;  Virtual`

Visibility: public

Description: `Add` adds `S` at the end of the list and returns the index of `S` in the list (which should equal Tstrings.Count (361)

See also: TStrings.Items (349), TStrings.AddObject (351), TStrings.Insert (356), TStrings.Delete (353), TStrings.Strings (363), TStrings.Count (361)

2.59.6 TStrings.AddObject

Synopsis: Add a string and associated object to the list.

Declaration: `function AddObject(const S: String;AObject: TObject) : Integer;  Virtual`

Visibility: public

Description: `AddObject` adds `S` to the list of strings, and associates `AObject` with it. It returns the index of `S`.

Remark: An object added to the list is not automatically destroyed by the list of the list is destroyed or the string it is associated with is deleted. It is the responsibility of the application to destroy any objects associated with strings.

See also: TStrings.Add (351), Tstrings.Items (349), TStrings.Objects (362), Tstrings.InsertObject (356)

2.59.7 TStrings.Append

Synopsis: Add a string to the list.

Declaration: `procedure Append(const S: String)`

Visibility: public

Description: `Append` does the same as TStrings.Add (351), only it does not return the index of the inserted string.

See also: TStrings.Add (351)

2.59.8 TStrings.AddStrings

Synopsis: Add contents of another stringlist to this list.

Declaration: `procedure AddStrings(TheStrings: TStrings); Virtual`

Visibility: public

Description: `AddStrings` adds the contents of `TheStrings` to the stringlist. Any associated objects are added as well.

See also: TStrings.Add (351), TStrings.Assign (352)

2.59.9 TStrings.Assign

Synopsis: Assign the contents of another stringlist to this one.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: public

Description: `Assign` replaces the contents of the stringlist with the contents of `Source` if `Source` is also of type `TStrings`. Any associated objects are copied as well.

See also: TStrings.Add (351), TStrings.AddStrings (352), TPersistent.Assign (317)

2.59.10 TStrings.BeginUpdate

Synopsis: Mark the beginning of an update batch.

Declaration: `procedure BeginUpdate`

Visibility: public

Description: `BeginUpdate` increases the update count by one. It is advisable to call `BeginUpdate` before lengthy operations on the stringlist. At the end of these operation, TStrings.EndUpdate (354) should be called to mark the end of the operation. Descendent classes may use this information to perform optmizations. e.g. updating the screen only once after many strings were added to the list.

All `TStrings` methods that modify the string list call `BeginUpdate` before the actual operation, and call `endUpdate` when the operation is finished. Descendent classes should also call these methods when modifying the string list.

Remark: Always put the corresponding call to TStrings.EndUpdate (354) in the context of a `Finally` block, to ensure that the update count is always descreased at the end of the operation, even if an exception occurred:

```
With MyStrings do
    try
      BeginUpdate;
      // Some lengthy operation.
    Finally
      EndUpdate
    end;
```

See also: TStrings.EndUpdate (354)

2.59.11 TStrings.Clear

Synopsis: Removes all strings and associated objects from the list.

Declaration: `procedure Clear;  Virtual;  Abstract`

Visibility: public

Description: `Clear` will remove all strings and their associated objects from the list. After a call to clear, TStrings.Count (361) is zero.

Since it is an abstract method, `TStrings` itself does not implement `Clear`. Descendent classes such as TStringList (345) implement this method.

See also: TStrings.Items (349), TStrings.Delete (353), TStrings.Count (361)

2.59.12 TStrings.Delete

Synopsis: Delete a string from the list.

Declaration: `procedure Delete(Index: Integer);  Virtual;  Abstract`

Visibility: public

Description: `Delete` deletes the string at position `Index` from the list. The associated object is also removed from the list, but not destroyed. `Index` is zero-based, and should be in the range `0` to `Count-1`.

Since it is an abstract method, `TStrings` itself does not implement `Delete`. Descendent classes such as TStringList (345) implement this method.

Errors: If `Index` is not in the allowed range, an EStringListError (217) is raised.

See also: TStrings.Insert (356), TStrings.Items (349), TStrings.Clear (353)

2.59.13 TStrings.EndUpdate

Synopsis: Mark the end of an update batch.

Declaration: `procedure EndUpdate`

Visibility: public

Description: `EndUpdate` should be called at the end of a lengthy operation on the stringlist, but only if there was a call to `BeginUpdate` before the operation was started. It is best to put the call to `EndUpdate` in the context of a `Finally` block, so it will be called even if an exception occurs.

For more information, see TStrings.BeginUpdate (352).

See also: TStrings.BeginUpdate (352)

2.59.14 TStrings.Equals

Synopsis: Compares the contents of two stringlists.

Declaration: `function Equals(TheStrings: TStrings) : Boolean`

Visibility: public

Description: `Equals` compares the contents of the stringlist with the contents of `TheStrings`. If the contents match, i.e. the stringlist contain an equal amount of strings, and all strings match, then `True` is returned. If the number of strings in the lists is is unequal, or they contain one or more different strings, `False` is returned.

Remark:

1.The strings are compared case-insensitively.

2.The associated objects are not compared

See also: Tstrings.Items (349), TStrings.Count (361), TStrings.Assign (352)

2.59.15 TStrings.Exchange

Synopsis: Exchanges two strings in the list.

Declaration: `procedure Exchange(Index1: Integer;Index2: Integer);  Virtual`

Visibility: public

Description: `Exchange` exchanges the strings at positions `Index1` and `Index2`. The associated objects are also exchanged.

Both indexes must be in the range of valid indexes, i.e. must have a value between `0` and `Count-1`.

Errors: If either `Index1` or `Index2` is not in the range of valid indexes, an EStringListError (217) exception is raised.

See also: TStrings.Move (357), TStrings.Strings (363), TStrings.Count (361)

2.59.16 TStrings.GetText

Synopsis: Returns the contents as a PChar

Declaration: `function GetText : PChar;  Virtual`

Visibility: public

Description: `GetText` allocates a memory buffer and compies the contents of the stringlist to this buffer as a series of strings, separated by an end-of-line marker. The buffer is zero terminated.

Remark: The caller is responsible for freeing the returned memory buffer.

2.59.17 TStrings.IndexOf

Synopsis: Find a string in the list and return its position.

Declaration: `function IndexOf(const S: String) : Integer;  Virtual`

Visibility: public

Description: `IndexOf` searches the list for `S`. The search is case-insensitive. If a matching entry is found, its position is returned. if no matching string is found, `-1` is returned.

Remark:

1.Only the first occurrence of the string is returned.

2.The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: TStrings.IndexOfObject (355), TStrings.IndexOfName (355), TStrings.Strings (363)

2.59.18 TStrings.IndexOfName

Synopsis: Finds the index of a name in the name-value pairs.

Declaration: `function IndexOfName(const Name: String) : Integer;  Virtual`

Visibility: public

Description: `IndexOfName` searches in the list of strings for a name-value pair with name part `Name`. If such a pair is found, it returns the index of the pair in the stringlist. If no such pair is found, the function returns `-1`. The search is done case-insensitive.

Remark:

1.Only the first occurrence of a matching name-value pair is returned.

2.The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: TStrings.IndexOf (355), TStrings.IndexOfObject (355), TStrings.Strings (363)

2.59.19 TStrings.IndexOfObject

Synopsis: Finds an object in the list and returns its index.

Declaration: `function IndexOfObject(AObject: TObject) : Integer;  Virtual`

Visibility: public

Description: `IndexOfObject` searchs through the list of strings till it find a string associated with `AObject`, and returns the index of this string. If no such string is found, `-1` is returned.

Remark:

1. Only the first occurrence of a string with associated object `AObject` is returned; if more strings in the list can be associated with `AObject`, they will not be found by this routine.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

2.59.20 TStrings.Insert

Synopsis: Insert a string in the list.

Declaration: `procedure Insert(Index: Integer;const S: String); Virtual; Abstract`

Visibility: public

Description: `Insert` inserts the string `S` at position `Index` in the list. `Index` is a zero-based position, and can have values from `0` to `Count`. If `Index` equals `Count` then the string is appended to the list.

Remark:

1. All methods that add strings to the list use `Insert` to add a string to the list.
2. If the string has an associated object, use TStrings.InsertObject (356) instead.

Errors: If `Index` is less than zero or larger than `Count` then an EStringListError (217) exception is raised.

See also: TStrings.Add (351), TStrings.InsertObject (356), TStrings.Append (352), TStrings.Delete (353)

2.59.21 TStrings.InsertObject

Synopsis: Insert a string and associated object in the list.

Declaration: `procedure InsertObject(Index: Integer;const S: String;AObject: TObject)`

Visibility: public

Description: `InsertObject` inserts the string `S` and its associated object`AObject` at position `Index` in the list. `Index` is a zero-based position, and can have values from `0` to `Count`. If `Index` equals `Count` then the string is appended to the list.

Errors: If `Index` is less than zero or larger than `Count` then an EStringListError (217) exception is raised.

See also: TStrings.Insert (356), TStrings.AddObject (351), TStrings.Append (352), TStrings.Delete (353)

2.59.22 TStrings.LoadFromFile

Synopsis: Load the contents of a file as a series of strings.

Declaration: `procedure LoadFromFile(const FileName: String); Virtual`

Visibility: public

Description: `LoadFromFile` loads the contents of a file into the stringlist. Each line in the file (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

`LoadFromFile` simply creates a file stream (284) with the given filename, and then executes TStrings.LoadfromStream (357); after that the file stream object is destroyed again.

See also: TStrings.LoadFromStream (357), TStrings.SaveToFile (358), Tstrings.SaveToStream (358)

2.59.23 TStrings.LoadFromStream

Synopsis: Load the contents of a stream as a series of strings.

Declaration: `procedure LoadFromStream(Stream: TStream); Virtual`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the stringlist. Each line in the stream (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

See also: TStrings.LoadFromFile (356), TStrings.SaveToFile (358), Tstrings.SaveToStream (358)

2.59.24 TStrings.Move

Synopsis: Move a string from one place in the list to another.

Declaration: `procedure Move(CurIndex: Integer;NewIndex: Integer); Virtual`

Visibility: public

Description: `Move` moves the string at position `CurIndex` so it has position `NewIndex` after the move operation. The object associated to the string is also moved. `CurIndex` and `NewIndex` should be in the range of `0` to `Count-1`.

Remark: `NewIndex` is *not* the position in the stringlist before the move operation starts. The move operation

1. removes the string from position `CurIndex`
2. inserts the string at position `NewIndex`

This may not lead to the desired result if `NewIndex` is bigger than `CurIndex`. Consider the following example:

```
With MyStrings do
    begin
    Clear;
    Add('String 0');
    Add('String 1');
    Add('String 2');
    Add('String 3');
    Add('String 4');
    Move(1,3);
    end;
```

After the `Move` operation has completed, 'String 1' will be between 'String 3' and 'String 4'.

Errors: If either `CurIndex` or `NewIndex` is outside the allowed range, an EStringListError (217) is raised.

See also: TStrings.Exchange (354)

2.59.25 TStrings.SaveToFile

Synopsis: Save the contents of the list to a file.

Declaration:
```
procedure SaveToFile(const FileName: String);  Virtual
```

Visibility: public

Description: `SaveToFile` saves the contents of the stringlist to the file with name `FileName`. It writes the strings to the file, separated by end-of-line markers, so each line in the file will contain 1 string from the stringlist.

`SaveToFile` creates a file stream (284) with name `FileName`, calls TStrings.SaveToStream (358) and then destroys the file stream object.

Errors: An EStreamError (216) exception can be raised if the file `FileName` cannot be opened, or if it cannot be written to.

See also: TStrings.SaveToStream (358), Tstrings.LoadFromStream (357), TStrings.LoadFromFile (356)

2.59.26 TStrings.SaveToStream

Synopsis: Save the contents of the string to a stream.

Declaration:
```
procedure SaveToStream(Stream: TStream);  Virtual
```

Visibility: public

Description: `SaveToStream` saves the contents of the stringlist to `Stream`. It writes the strings to the stream, separated by end-of-line markers, so each 'line' in the stream will contain 1 string from the stringlist.

Errors: An EStreamError (216) exception can be raised if the stream cannot be written to.

See also: TStrings.SaveToFile (358), Tstrings.LoadFromStream (357), TStrings.LoadFromFile (356)

2.59.27 TStrings.SetText

Synopsis: Set the contents of the list from a PChar.

Declaration:
```
procedure SetText(TheText: PChar);  Virtual
```

Visibility: public

Description: `SetText` parses the contents of `TheText` and fills the stringlist based on the contents. It regards `TheText` as a series of strings, separated by end-of-line markers. Each of these strings is added to the stringlist.

See also: TStrings.Text (363)

2.59.28 TStrings.GetNameValue

Synopsis: Return both name and value of a name,value pair based on it's index.

Declaration:
```
procedure GetNameValue(Index: Integer;out AName: String;
                       out AValue: String)
```

Visibility: public

Description: Return both name and value of a name,value pair based on it's index.

2.59.29 TStrings.ExtractName

Synopsis: Extract the name part of a string

Declaration: `function ExtractName(const S: String) : String`

Visibility: public

Description: `ExtractName` returns the name part (the part before the NameValueSeparator (360) character) of the string. If the character is not present, an empty string is returned. The resulting string is not trimmed, it can end or start with spaces.

See also: TStrings.NameValueSeparator (360)

2.59.30 TStrings.TextLineBreakStyle

Synopsis: Determines which line breaks to use in the Text (363) property

Declaration: `Property TextLineBreakStyle : TTextLineBreakStyle`

Visibility: public

Access: Read,Write

Description: `TextLineBreakStyle` determines which linebreak style is used when constructing the Text property: the same rules are used as in the writing to text files:

tlbsLFLines are separated with a linefeed character #10.

tlbsCRLFLines are separated with a carriage-return/linefeed character pair: #13#10.

tlbsCRLines are separated with a carriage-return character #13.

It has no effect when setting the text property.

See also: #rtl.classes.TStrings.Text (363)

2.59.31 TStrings.Delimiter

Synopsis: Delimiter character used in DelimitedText (359).

Declaration: `Property Delimiter : Char`

Visibility: public

Access: Read,Write

Description: Delimiter character used in DelimitedText (359).

2.59.32 TStrings.DelimitedText

Synopsis: Get or set all strings in the list in a delimited form.

Declaration: `Property DelimitedText : String`

Visibility: public

Access: Read,Write

Description: Get or set all strings in the list in a delimited form.

2.59.33 TStrings.QuoteChar

Synopsis: Quote character used in DelimitedText (359).

Declaration: `Property QuoteChar : Char`

Visibility: public

Access: Read,Write

Description: Quote character used in DelimitedText (359).

2.59.34 TStrings.NameValueSeparator

Synopsis: Value of the character used to separate name,value pairs

Declaration: `Property NameValueSeparator : Char`

Visibility: public

Access: Read,Write

Description: Value of the character used to separate name,value pairs

2.59.35 TStrings.ValueFromIndex

Synopsis:

Declaration: `Property ValueFromIndex[Index: Integer]: String`

Visibility: public

Access: Read,Write

Description: Return the value part of a string based on it's index.

2.59.36 TStrings.Capacity

Synopsis: Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of strings that the list can hold before it tries to allocate more memory.

`TStrings` returns TStrings.Count (361) when read. Trying to set the capacity has no effect. Descendent classes such as TStringlist (345) can override this property such that it actually sets the new capacity.

See also: TStringList (345), TStrings.Count (361)

2.59.37 TStrings.CommaText

Synopsis: Contents of the list as a comma-separated string.

Declaration: `Property CommaText : String`

Visibility: public

Access: Read,Write

Description: `CommaText` represents the stringlist as a single string, consisting of a comma-separated concatenation of the strings in the list. If one of the strings contains spaces, comma's or quotes it will be enclosed by double quotes. Any double quotes in a string will be doubled. For instance the following strings:

```
Comma,string
Quote"string
Space string
NormalSttring
```

is converted to

```
"Comma,string","Quote""String","Space string",NormalString
```

Conversely, when setting the `CommaText` property, the text will be parsed according to the rules outlined above, and the strings will be set accordingly. Note that spaces will in this context be regarded as string separators, unless the string as a whole is contained in double quotes. Spaces that occur next to a delimiter will be ignored. The following string:

```
"Comma,string" , "Quote""String",Space string,, NormalString
```

Will be converted to

```
Comma,String
Quote"String
Space
String

NormalString
```

This is a special case of the DelimitedText (185) property where the quote character is always the double quote, and the delimiter is always the colon.

See also: TStrings.Text (363), TStrings.SetText (358)

2.59.38 TStrings.Count

Synopsis: Number of strings in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the current number of strings in the list. `TStrings` does not implement this property; descendent classes should override the property read handler to return the correct value.

Strings in the list are always uniquely identified by their `Index`; the index of a string is zero-based, i.e. it's supported range is `0` to `Count-1`. trying to access a string with an index larger than or equal to `Count` will result in an error. Code that iterates over the list in a stringlist should always take into account the zero-based character of the list index.

See also: TStrings.Strings (363), TStrings.Objects (362), TStrings.Capacity (360)

2.59.39 TStrings.Names

Synopsis: Name parts of the name-value pairs in the list.

Declaration: `Property Names[Index: Integer]: String`

Visibility: public

Access: Read

Description: `Names` provides indexed access to the names of the name-value pairs in the list. It returns the name part of the `Index`-th string in the list.

Remark: The index is not an index based on the number of name-value pairs in the list. It is the name part of the name-value pair a string `Index` in the list. If the string at position `Index` is not a name-value pair (i.e. does not contain the equal sign (=)), then an empty name is returned.

See also: TStrings.Values (362), TStrings.IndexOfName (355)

2.59.40 TStrings.Objects

Synopsis: Indexed access to the objects associated with the strings in the list.

Declaration: `Property Objects[Index: Integer]: TObject`

Visibility: public

Access: Read,Write

Description: `Objects` provides indexed access to the objects associated to the strings in the list. `Index` is a zero-based index and must be in the range of `0` to `Count-1`.

Setting the `objects` property will not free the previously associated object, if there was one. The caller is repsonsible for freeing the object that was previously associated to the string.

`TStrings` does not implement any storage for objects. Reading the `Objects` property will always return `Nil`, Setting the property will have no effect. It is the responsability of the descendent classes to provide storage for the associated objects.

Errors: If an `Index` outside the valid range is specified, an EStringListError (217) exception will be raised.

See also: TStrings.Strings (363), TStrings.IndexOfObject (355), TStrings.Names (362), TStrings.Values (362)

2.59.41 TStrings.Values

Synopsis: Value parts of the name-value pairs in the list.

Declaration: `Property Values[Name: String]: String`

Visibility: public

Access: Read,Write

Description: `Values` represents the value parts of the name-value pairs in the list.

When reading this property, if there is a name-value pair in the list of strings that has name part `Name`, then the corresponding value is returned. If there is no such pair, an empty string is returned.

When writing this value, first it is checked whether there exists a name-value pair in the list with name `Name`. If such a pair is found, it's value part is overwritten with the specified value. If no such pair is found, a new name-value pair is added with the specified `Name` and value.

Remark:

1. Names are compared case-insensitively.
2. Any character, including whitespace, up till the first equal (=) sign in a string is considered part of the name.

See also: TStrings.Names (362), TStrings.Strings (363), TStrings.Objects (362)

2.59.42 TStrings.Strings

Synopsis: Indexed access to the strings in the list.

Declaration: `Property Strings[Index: Integer]: String; default`

Visibility: public

Access: Read,Write

Description: `Strings` is the default property of `TStrings`. It provides indexed read-write access to the list of strings. Reading it will return the string at position `Index` in the list. Writing it will set the string at position `Index`.

`Index` is the position of the string in the list. It is zero-based, i.e. valued values range from `0` (the first string in the list) till `Count-1` (the last string in the list). When browsing through the strings in the list, this fact must be taken into account.

To access the objects associated with the strings in the list, use the TStrings.Objects (362) property. The name parts of name-value pairs can be accessed with the TStrings.Names (362) property, and the values can be set or read through the TStrings.Values (362) property.

Searching through the list can be done using the TStrings.IndexOf (355) method.

Errors: If `Index` is outside the allowed range, an EStringListError (217) exception is raised.

See also: TStrings.Count (361), TStrings.Objects (362), TStrings.Names (362), TStrings.Values (362), TStrings.IndexOf (355)

2.59.43 TStrings.Text

Synopsis: Contents of the list as one big string.

Declaration: `Property Text : String`

Visibility: public

Access: Read,Write

Description: `Text` returns, when read, the contents of the stringlist as one big string consisting of all strings in the list, separated by an end-of-line marker. When this property is set, the string will be cut into smaller strings, based on the positions of end-of-line markers in the string. Any previous content of the stringlist will be lost.

Remark: If any of the strings in the list contains an end-of-line marker, then the resulting string will appear to contain more strings than actually present in the list. To avoid this ambiguity, use the TStrings.CommaText (361) property instead.

See also: TStrings.Strings (363), TStrings.Count (361), TStrings.CommaText (361)

2.59.44 TStrings.StringsAdapter

Synopsis: Not implemented in Free Pascal.

Declaration: `Property StringsAdapter : IStringsAdapter`

Visibility: public

Access: Read,Write

Description: Not implemented in Free Pascal.

2.60 TStringStream

2.60.1 Description

`TStringStream` stores its data in an ansistring. The contents of this string is available as the DataString (366) property. It also introduces some methods to read or write parts of the stringstream's data as a string.

The main purpose of a `TStringSTream` is to be able to treat a string as a stream from which can be read.

2.60.2 Method overview

Page	Property	Description
364	Create	Creates a new stringstream and sets its initial content.
365	Read	Reads from the stream.
365	ReadString	Reads a string of length `Count`
365	Seek	Sets the position in the stream.
365	Write	`Write` overries the TStream.Write (333) method.
366	WriteString	`WriteString` writes a string to the stream.

2.60.3 Property overview

Page	Property	Access	Description
366	DataString	r	Contains the contents of the stream in string form

2.60.4 TStringStream.Create

Synopsis: Creates a new stringstream and sets its initial content.

Declaration: `constructor Create(const AString: String)`

Visibility: public

Description: `Create` creates a new `TStringStream` instance and sets its initial content to `Astring`. The position is still 0 but the size of the stream will equal the length of the string.

See also: TStringStream.DataString (366)

2.60.5 TStringStream.Read

Synopsis: Reads from the stream.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` overrides the Read (333) from TStream (332). It tries to read `Count` bytes into `Buffer`. It returns the number of bytes actually read. The position of the stream is advanced with the number of bytes actually read; When the reading has reached the end of the DataString (366), then the reading stops, i.e. it is not possible to read beyond the end of the datastring.

See also: TStream.Read (333), TStringStream.Write (365), TStringStream.DataString (366)

2.60.6 TStringStream.ReadString

Synopsis: Reads a string of length `Count`

Declaration: `function ReadString(Count: LongInt) : String`

Visibility: public

Description: `ReadString` reads `Count` bytes from the stream and returns the read bytes as a string. If less than `Count` bytes were available, the string has as many characters as bytes could be read.

The `ReadString` method is a wrapper around the Read (365) method. It does not do the same stringas the TStream.ReadAnsiString (338) method, which first reads a length integer to determine the length of the string to be read.

See also: TStringStream.Read (365), TStream.ReadAnsiString (338)

2.60.7 TStringStream.Seek

Synopsis: Sets the position in the stream.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` implements the abstract Seek (333) method.

2.60.8 TStringStream.Write

Synopsis: `Write` overries the TStream.Write (333) method.

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` overries the TStream.Write (333) method.

2.60.9 TStringStream.WriteString

Synopsis: `WriteString` writes a string to the stream.

Declaration: `procedure WriteString(const AString: String)`

Visibility: public

Description: `WriteString` writes a string to the stream.

2.60.10 TStringStream.DataString

Synopsis: Contains the contents of the stream in string form

Declaration: `Property DataString : String`

Visibility: public

Access: Read

Description: Contains the contents of the stream in string form

2.61 TTextObjectWriter

2.61.1 Description

Not yet implemented.

2.62 TThread

2.62.1 Description

The `TThread` class encapsulates the native thread support of the operating system. To create a thread, declare a descendent of the `TThread` object and override the Execute (366) method. In this method, the tthread's code should be executed. To run a thread, create an instance of the tthread descendent, and call it's execute method.

2.62.2 Method overview

Page	Property	Description
367	AfterConstruction	Code to be executed after construction but before execute.
367	Create	Creates a new thread.
367	Destroy	Destroys the thread object.
367	Resume	Resumes the thread's execution.
368	Suspend	Suspends the thread's execution.
368	Terminate	Signals the thread it should terminate.
368	WaitFor	Waits for the thread to terminate and returns the exit status.

2.62.3 Property overview

Page	Property	Access	Description
369	FatalException	r	Exception that occured during thread execution
368	FreeOnTerminate	rw	Indicates whether the thread should free itself when it stops executing.
368	Handle	r	Returns the thread handle.
369	OnTerminate	rw	Event called when the thread terminates.
369	Priority	rw	Returns the thread priority.
369	Suspended	rw	Indicates whether the thread is suspended.
369	ThreadID	r	Returns the thread ID.

2.62.4 TThread.Create

Synopsis: Creates a new thread.

Declaration: `constructor Create(CreateSuspended: Boolean;const StackSize: SizeUInt)`

Visibility: public

Description: Creates a new thread.

2.62.5 TThread.Destroy

Synopsis: Destroys the thread object.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the thread object.

2.62.6 TThread.AfterConstruction

Synopsis: Code to be executed after construction but before execute.

Declaration: `procedure AfterConstruction;  Override`

Visibility: public

Description: `AfterConstruction` is overridden in `TThread` but currently does not do anything useful.

2.62.7 TThread.Resume

Synopsis: Resumes the thread's execution.

Declaration: `procedure Resume`

Visibility: public

Description: Resumes the thread's execution.

2.62.8 TThread.Suspend

Synopsis: Suspends the thread's execution.

Declaration: `procedure Suspend`

Visibility: public

Description: Suspends the thread's execution.

2.62.9 TThread.Terminate

Synopsis: Signals the thread it should terminate.

Declaration: `procedure Terminate`

Visibility: public

Description: Signals the thread it should terminate.

2.62.10 TThread.WaitFor

Synopsis: Waits for the thread to terminate and returns the exit status.

Declaration: `function WaitFor : Integer`

Visibility: public

Description: Waits for the thread to terminate and returns the exit status.

2.62.11 TThread.FreeOnTerminate

Synopsis: Indicates whether the thread should free itself when it stops executing.

Declaration: `Property FreeOnTerminate : Boolean`

Visibility: public

Access: Read,Write

Description: Indicates whether the thread should free itself when it stops executing.

2.62.12 TThread.Handle

Synopsis: Returns the thread handle.

Declaration: `Property Handle : TThreadID`

Visibility: public

Access: Read

Description: Returns the thread handle.

2.62.13 TThread.Priority

Synopsis: Returns the thread priority.

Declaration: `Property Priority : TThreadPriority`

Visibility: public

Access: Read,Write

Description: Returns the thread priority.

2.62.14 TThread.Suspended

Synopsis: Indicates whether the thread is suspended.

Declaration: `Property Suspended : Boolean`

Visibility: public

Access: Read,Write

Description: Indicates whether the thread is suspended.

2.62.15 TThread.ThreadID

Synopsis: Returns the thread ID.

Declaration: `Property ThreadID : TThreadID`

Visibility: public

Access: Read

Description: Returns the thread ID.

2.62.16 TThread.OnTerminate

Synopsis: Event called when the thread terminates.

Declaration: `Property OnTerminate : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event called when the thread terminates.

2.62.17 TThread.FatalException

Synopsis: Exception that occured during thread execution

Declaration: `Property FatalException : TObject`

Visibility: public

Access: Read

Description: `FatalException` contains the exception that occurred during the thread's execution.

2.63 TThreadList

2.63.1 Description

This class is not yet implemented in Free Pascal.

2.63.2 Method overview

Page	Property	Description
370	Add	Adds an element to the list.
371	Clear	Removes all emements from the list.
370	Create	Creates a new thread-safe list.
370	Destroy	Destroys the list instance.
371	LockList	Locks the list for exclusive access.
371	Remove	Removes an item from the list.
371	UnlockList	Unlocks the list after it was locked.

2.63.3 Property overview

Page	Property	Access	Description
371	Duplicates	rw	Describes what to do with duplicates

2.63.4 TThreadList.Create

Synopsis: Creates a new thread-safe list.

Declaration: `constructor Create`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

2.63.5 TThreadList.Destroy

Synopsis: Destroys the list instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

2.63.6 TThreadList.Add

Synopsis: Adds an element to the list.

Declaration: `procedure Add(Item: Pointer)`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

2.63.7 TThreadList.Clear

Synopsis: Removes all emements from the list.

Declaration: `procedure Clear`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

2.63.8 TThreadList.LockList

Synopsis: Locks the list for exclusive access.

Declaration: `function LockList : TList`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

2.63.9 TThreadList.Remove

Synopsis: Removes an item from the list.

Declaration: `procedure Remove(Item: Pointer)`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

2.63.10 TThreadList.UnlockList

Synopsis: Unlocks the list after it was locked.

Declaration: `procedure UnlockList`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

2.63.11 TThreadList.Duplicates

Synopsis: Describes what to do with duplicates

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` describes what the threadlist should do when a duplicate pointer is added to the list. It is identical in behaviour to the Duplicates (348) property of TStringList (345).

See also: TDuplicates (190)

2.64 TWriter

2.64.1 Description

Object to write component data to an arbitrary format.

2.64.2 Method overview

Page	Property	Description
372	Create	Creates a new Writer with a stream and bufsize.
373	DefineBinaryProperty	Callback used when defining and streaming custom properties.
373	DefineProperty	Callback used when defining and streaming custom properties.
373	Destroy	Destroys the writer instance.
373	Write	Write raw data to stream
373	WriteBoolean	Write boolean value to the stream.
374	WriteChar	Write a character to the stream.
374	WriteCollection	Write a collection to the stream.
374	WriteComponent	Stream a component to the stream.
375	WriteCurrency	Write a currency value to the stream
375	WriteDate	Write a date to the stream.
374	WriteDescendent	Write a descendent component to the stream.
374	WriteFloat	Write a float to the stream.
375	WriteIdent	Write an identifier to the stream.
375	WriteInteger	Write an integer to the stream.
376	WriteListBegin	Write a start-of-list marker to the stream.
376	WriteListEnd	Write an end-of-list marker to the stream.
376	WriteRootComponent	Write a root component to the stream.
375	WriteSingle	Write a single-type real to the stream.
376	WriteString	Write a string to the stream.
374	WriteWideChar	Write widechar to stream
376	WriteWideString	Write a widestring value to the stream

2.64.3 Property overview

Page	Property	Access	Description
378	Driver	r	Driver used when writing to the stream.
377	OnFindAncestor	rw	Event occurring when an ancestor component must be found.
377	OnWriteMethodProperty	rw	Handler from writing method properties.
377	OnWriteStringProperty	rw	Event handler for translating strings written to stream.
378	PropertyPath	r	Path to the property that is currently being written
377	RootAncestor	rw	Ancestor of root component.

2.64.4 TWriter.Create

Synopsis: Creates a new Writer with a stream and bufsize.

Declaration:
```
constructor Create(ADriver: TAbstractObjectWriter)
constructor Create(Stream: TStream;BufSize: Integer)
```

Visibility: public

Description: Creates a new Writer with a stream and bufsize.

2.64.5 TWriter.Destroy

Synopsis: Destroys the writer instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroys the writer instance.

2.64.6 TWriter.DefineProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:
```
procedure DefineProperty(const Name: String;ReadData: TReaderProc;
                         AWriteData: TWriterProc;HasData: Boolean)
                         ;  Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

2.64.7 TWriter.DefineBinaryProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:
```
procedure DefineBinaryProperty(const Name: String;ReadData: TStreamProc;
                               AWriteData: TStreamProc;HasData: Boolean)
                               ;  Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

2.64.8 TWriter.Write

Synopsis: Write raw data to stream

Declaration:
```
procedure Write(const Buffer;Count: LongInt);  Virtual
```

Visibility: public

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TBinaryObjectWriter.Write (251), TAbstractObjectWriter.Write (235)

2.64.9 TWriter.WriteBoolean

Synopsis: Write boolean value to the stream.

Declaration:
```
procedure WriteBoolean(Value: Boolean)
```

Visibility: public

Description: Write boolean value to the stream.

2.64.10 TWriter.WriteCollection

Synopsis: Write a collection to the stream.

Declaration: `procedure WriteCollection(Value: TCollection)`

Visibility: public

Description: Write a collection to the stream.

2.64.11 TWriter.WriteComponent

Synopsis: Stream a component to the stream.

Declaration: `procedure WriteComponent(Component: TComponent)`

Visibility: public

Description: Stream a component to the stream.

2.64.12 TWriter.WriteChar

Synopsis: Write a character to the stream.

Declaration: `procedure WriteChar(Value: Char)`

Visibility: public

Description: Write a character to the stream.

2.64.13 TWriter.WriteWideChar

Synopsis: Write widechar to stream

Declaration: `procedure WriteWideChar(Value: WideChar)`

Visibility: public

Description: `WriteWideChar` writes a widechar to the stream. This actually writes a widestring of length 1.

See also: TReader.ReadWideChar (323), TWriter.WriteWideString (376)

2.64.14 TWriter.WriteDescendent

Synopsis: Write a descendent component to the stream.

Declaration: `procedure WriteDescendent(ARoot: TComponent;AAncestor: TComponent)`

Visibility: public

Description: Write a descendent component to the stream.

2.64.15 TWriter.WriteFloat

Synopsis: Write a float to the stream.

Declaration: `procedure WriteFloat(const Value: Extended)`

Visibility: public

Description: Write a float to the stream.

2.64.16 TWriter.WriteSingle

Synopsis: Write a single-type real to the stream.

Declaration:
```
procedure WriteSingle(const Value: Single)
```

Visibility: public

Description: Write a single-type real to the stream.

2.64.17 TWriter.WriteCurrency

Synopsis: Write a currency value to the stream

Declaration:
```
procedure WriteCurrency(const Value: Currency)
```

Visibility: public

Description: `WriteCurrency` writes a currency typed value to the stream. This method does nothing except call the driver method of the driver being used.

See also: TReader.ReadCurrency (324)

2.64.18 TWriter.WriteDate

Synopsis: Write a date to the stream.

Declaration:
```
procedure WriteDate(const Value: TDateTime)
```

Visibility: public

Description: Write a date to the stream.

2.64.19 TWriter.WriteIdent

Synopsis: Write an identifier to the stream.

Declaration:
```
procedure WriteIdent(const Ident: String)
```

Visibility: public

Description: Write an identifier to the stream.

2.64.20 TWriter.WriteInteger

Synopsis: Write an integer to the stream.

Declaration:
```
procedure WriteInteger(Value: LongInt);  Overload
procedure WriteInteger(Value: Int64);  Overload
```

Visibility: public

Description: Write an integer to the stream.

2.64.21 TWriter.WriteListBegin

Synopsis: Write a start-of-list marker to the stream.

Declaration: `procedure WriteListBegin`

Visibility: public

Description: Write a start-of-list marker to the stream.

2.64.22 TWriter.WriteListEnd

Synopsis: Write an end-of-list marker to the stream.

Declaration: `procedure WriteListEnd`

Visibility: public

Description: Write an end-of-list marker to the stream.

2.64.23 TWriter.WriteRootComponent

Synopsis: Write a root component to the stream.

Declaration: `procedure WriteRootComponent(ARoot: TComponent)`

Visibility: public

Description: Write a root component to the stream.

2.64.24 TWriter.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString(const Value: String)`

Visibility: public

Description: Write a string to the stream.

2.64.25 TWriter.WriteWideString

Synopsis: Write a widestring value to the stream

Declaration: `procedure WriteWideString(const Value: WideString)`

Visibility: public

Description: `WriteWidestring` writes a currency typed value to the stream. This method does nothing except call the driver method of the driver being used.

See also: TReader.ReadWideString (326)

2.64.26 TWriter.RootAncestor

Synopsis: Ancestor of root component.

Declaration: `Property RootAncestor : TComponent`

Visibility: public

Access: Read,Write

Description: Ancestor of root component.

2.64.27 TWriter.OnFindAncestor

Synopsis: Event occurring when an ancestor component must be found.

Declaration: `Property OnFindAncestor : TFindAncestorEvent`

Visibility: public

Access: Read,Write

Description: Event occurring when an ancestor component must be found.

2.64.28 TWriter.OnWriteMethodProperty

Synopsis: Handler from writing method properties.

Declaration: `Property OnWriteMethodProperty : TWriteMethodPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnWriteMethodProperty` can be set by an IDE or some streaming mechanism which handles dummy values for method properties; It can be used to write a real value to the stream which will be interpreted correctly when the stream is read. See TWriteMethodPropertyEvent (198) for a description of the arguments.

See also: TWriteMethodPropertyEvent (198), TReader.OnSetMethodProperty (328)

2.64.29 TWriter.OnWriteStringProperty

Synopsis: Event handler for translating strings written to stream.

Declaration: `Property OnWriteStringProperty : TReadWriteStringPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnWriteStringProperty` is called whenever a string property is written to the stream. It can be used e.g. by a translation mechanism to translate the strings on the fly, when a form is written. See TReadWriteStringPropertyEvent (195) for a description of the various parameters.

See also: TReader.OnPropertyNotFound (327), TReader.OnSetMethodProperty (328), TReadWriteStringPropertyEvent (195)

2.64.30 TWriter.Driver

Synopsis: Driver used when writing to the stream.

Declaration: `Property Driver : TAbstractObjectWriter`

Visibility: public

Access: Read

Description: Driver used when writing to the stream.

2.64.31 TWriter.PropertyPath

Synopsis: Path to the property that is currently being written

Declaration: `Property PropertyPath : String`

Visibility: public

Access: Read

Description: `PropertyPath` is set to the property name of the class currently being written to stream. This is only done when TPersistent (317) descendent class properties are written.

Chapter 3

Reference for unit 'Crt'

3.1 Overview

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

There are some caveats when using the CRT unit:

- Programs using the CRT unit will *not* be usable when input/output is being redirected on the command-line.
- For similar reasons they are not usable as CGI-scripts for use with a webserver.
- The use of the CRT unit and the graph unit may not always be supported.
- On linux or other unix OSes , executing other programs that expect special terminal behaviour (using one of the special functions in the linux unit) will not work. The terminal is set in RAW mode, which will destroy most terminal emulation settings.

3.2 Constants, types and variables

3.2.1 Constants

```
Black = 0
```

Black color attribute

```
Blink = 128
```

Blink attribute

```
Blue = 1
```

Blue color attribute

```
Brown = 6
```

Brown color attribute

```
BW40 = 0
```

40 columns black and white screen mode.

```
BW80 = 2
```

80 columns black and white screen mode.

```
C40 = CO40
```

40 columns color screen mode.

```
C80 = CO80
```

80 columns color screen mode.

```
CO40 = 1
```

40 columns color screen mode.

```
CO80 = 3
```

80 columns color screen mode.

```
ConsoleMaxX = 1024
```

```
ConsoleMaxY = 1024
```

```
Cyan = 3
```

Cyan color attribute

```
DarkGray = 8
```

Dark gray color attribute

```
Flushing = false
```

```
Font8x8 = 256
```

Internal ROM font mode

```
Green = 2
```

Green color attribute

```
LightBlue = 9
```

Light Blue color attribute

```
LightCyan = 11
```

Light cyan color attribute

```
LightGray = 7
```

Light gray color attribute

```
LightGreen = 10
```

Light green color attribute

```
LightMagenta = 13
```

Light magenta color attribute

```
LightRed = 12
```

Light red color attribute

```
Magenta = 5
```

Magenta color attribute

```
Mono = 7
```

Monochrome screen mode (hercules screens)

```
Red = 4
```

Red color attribute

```
ScreenHeight : LongInt = 25
```

Current screen height.

```
ScreenWidth : LongInt = 80
```

Current screen width

```
White = 15
```

White color attribute

```
Yellow = 14
```

Yellow color attribute

3.2.2 Types

```
PConsoleBuf = ^TConsoleBuf
```

```
TCharAttr = packed record
  ch : Char;
  attr : Byte;
end
```

```
TConsoleBuf = Array[0..ConsoleMaxX*ConsoleMaxY-1] of TCharAttr
```

3.2.3 Variables

```
CheckBreak : Boolean
```

Check for CTRL-Break keystroke. Not used.

```
CheckEOF : Boolean
```

Check for EOF on standard input. Not used.

```
CheckSnow : Boolean
```

Check snow on CGA screens. Not used.

```
ConsoleBuf : PConsoleBuf
```

```
DirectVideo : Boolean
```

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.

```
LastMode : Word = 3
```

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

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

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

```
WindMax : Word = $184f
```

The upper byte of `WindMax` contains the Y coordinate while the lower byte contains the X coordinate. The use of this variable is deprecated, use `WindMaxX` and `WindMaxY` instead.

```
WindMaxX : DWord
```

X coordinate of lower right corner of the defined window

`WindMaxY : DWord`

Y coordinate of lower right corner of the defined window

`WindMin : Word = $0`

The upper byte of `WindMin` contains the Y coordinate while the lower byte contains the X coordinate. The use of this variable is deprecated, use `WindMinX` and `WindMinY` instead.

`WindMinX : DWord`

X coordinate of upper left corner of the defined window

`WindMinY : DWord`

Y coordinate of upper left corner of the defined window

3.3 Procedures and functions

3.3.1 AssignCrt

Synopsis: Assign file to CRT.

Declaration: `procedure AssignCrt(var F: Text)`

Visibility: default

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

Errors: None.

See also: Window (394)

Listing: ./crtex/ex1.pp

```
Program Example1;
uses Crt;

{ Program to demonstrate the AssignCrt function. }

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

3.3.2 ClrEol

Synopsis: Clear from cursor position till end of line.

Declaration: `procedure ClrEol`

Visibility: default

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 (386), InsLine (388), ClrScr (384)

Listing: ./crtex/ex9.pp

```
Program Example9;
uses Crt;

{ Program to demonstrate the ClrEol function. }
var
  I,J : integer;

begin
  For I:=1 to 15 do
    For J:=1 to 80 do
      begin
      gotoxy(j,i);
      Write(j mod 10);
      end;
  Window(5,5,75,12);
  Write('This line will be cleared from',
        ' here till the right of the window');
  GotoXY(27,WhereY);
  ReadKey;
  ClrEol;
  WriteLn;
end.
```

3.3.3 ClrScr

Synopsis: Clear current window.

Declaration: `procedure ClrScr`

Visibility: default

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 (394)

Listing: ./crtex/ex8.pp

```
Program Example8;
uses Crt;

{ Program to demonstrate the ClrScr function. }

begin
  WriteLn('Press any key to clear the screen');
  ReadKey;
  ClrScr;
  WriteLn('Have fun with the cleared screen');
end.
```

3.3.4 cursorbig

Synopsis: Show big cursor

Declaration: `procedure cursorbig`

Visibility: default

Description: `CursorBig` makes the cursor a big rectangle. Not implemented on unixes.

Errors: None.

See also: CursorOn (385), CursorOff (385)

3.3.5 cursoroff

Synopsis: Hide cursor

Declaration: `procedure cursoroff`

Visibility: default

Description: `CursorOff` switches the cursor off (i.e. the cursor is no longer visible). Not implemented on unixes.

Errors: None.

See also: CursorOn (385), CursorBig (385)

3.3.6 cursoron

Synopsis: Display cursor

Declaration: `procedure cursoron`

Visibility: default

Description: `CursorOn` switches the cursor on. Not implemented on unixes.

Errors: None.

See also: CursorBig (385), CursorOff (385)

3.3.7 Delay

Synopsis: Delay program execution.

Declaration: `procedure Delay(MS: Word)`

Visibility: default

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 (391), NoSound (390)

Listing: ./crtex/ex15.pp

```
Program Example15;
uses Crt;

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

3.3.8 DelLine

Synopsis: Delete line at cursor position.

Declaration: `procedure DelLine`

Visibility: default

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 (384), InsLine (388), ClrScr (384)

Listing: ./crtex/ex11.pp

```
Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
  ClrScr;
  WriteLn;
  WriteLn('Line 1');
```

```
  WriteLn('Line 2');
  WriteLn('Line 2');
  WriteLn('Line 3');
  WriteLn;
  WriteLn('Oops, Line 2 is listed twice,',
          ' let''s delete the line at the cursor postion');
  GotoXY(1,3);
  ReadKey;
  DelLine;
  GotoXY(1,10);
end.
```

3.3.9 GotoXY

Synopsis: Set cursor position on screen.

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

Visibility: default

Description: `GotoXY` 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 (393), WhereY (393), Window (394)

Listing: ./crtex/ex6.pp

```
Program Example6;
uses Crt;

{ Program to demonstrate the GotoXY function. }

begin
  ClrScr;
  GotoXY(10,10);
  Write('10,10');
  GotoXY(70,20);
  Write('70,20');
  GotoXY(1,22);
end.
```

3.3.10 HighVideo

Synopsis: Switch to highlighted text mode

Declaration: `procedure HighVideo`

Visibility: default

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

Errors: None.

See also: TextColor (392), TextBackground (391), LowVideo (389), NormVideo (389)

Listing: ./crtex/ex14.pp

```
Program Example14;
uses Crt;

{ Program to demonstrate the LowVideo, HighVideo, NormVideo functions. }

begin
  LowVideo;
  WriteLn('This is written with LowVideo');
  HighVideo;
  WriteLn('This is written with HighVideo');
  NormVideo;
  WriteLn('This is written with NormVideo');
end.
```

3.3.11 InsLine

Synopsis: Insert an empty line at cursor position

Declaration: `procedure InsLine`

Visibility: default

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 (384), DelLine (386), ClrScr (384)

Listing: ./crtex/ex10.pp

```
Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

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

3.3.12 KeyPressed

Synopsis: Check if there is a keypress in the keybuffer

Declaration: `function KeyPressed : Boolean`

Visibility: default

Description: `Keypressed` 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 (390)

Listing: ./crtex/ex2.pp

```
Program Example2;
uses Crt;

{ Program to demonstrate the KeyPressed function. }

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

3.3.13 LowVideo

Synopsis: Switch to low intensity colors.

Declaration: `procedure LowVideo`

Visibility: default

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

For an example, see HighVideo (387)

Errors: None.

See also: TextColor (392), TextBackground (391), HighVideo (387), NormVideo (389)

3.3.14 NormVideo

Synopsis: Return to normal (startup) modus

Declaration: `procedure NormVideo`

Visibility: default

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

For an example, see HighVideo (387)

Errors: None.

See also: TextColor (392), TextBackground (391), LowVideo (389), HighVideo (387)

3.3.15 NoSound

Synopsis: Stop system speaker

Declaration: `procedure NoSound`

Visibility: default

Description: `NoSound` stops the speaker sound. This call is not supported on all operating systems.

Errors: None.

See also: Sound (391)

Listing: ./crtex/ex16.pp

```
Program Example16;
uses Crt;

{ Program to demonstrate the Sound and NoSound function. }

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

3.3.16 ReadKey

Synopsis: Read key from keybuffer

Declaration: `function ReadKey : Char`

Visibility: default

Description: `ReadKey` 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.

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 (388)

Listing: ./crtex/ex3.pp

```
Program Example3;
uses Crt;

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

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

3.3.17 Sound

Synopsis: Sound system speaker

Declaration: `procedure Sound(Hz: Word)`

Visibility: default

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

Errors: None.

See also: NoSound (390)

3.3.18 TextBackground

Synopsis: Set text background

Declaration: `procedure TextBackground(Color: Byte)`

Visibility: default

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

Errors: None.

See also: TextColor (392), HighVideo (387), LowVideo (389), NormVideo (389)

Listing: ./crtex/ex13.pp

```
Program Example13;
uses Crt;

{ Program to demonstrate the TextBackground function. }

begin
```

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

3.3.19 TextColor

Synopsis: Set text color

Declaration: `procedure TextColor(Color: Byte)`

Visibility: default

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

Errors: None.

See also: TextBackground (391), HighVideo (387), LowVideo (389), NormVideo (389)

Listing: ./crtex/ex12.pp

```
Program Example12;
uses Crt;

{ Program to demonstrate the TextColor function. }

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

3.3.20 TextMode

Synopsis: Set screen mode.

Declaration: `procedure TextMode(Mode: Word)`

Visibility: default

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

This procedure is only implemented on dos.

Errors: None.

See also: Window (394)

3.3.21 WhereX

Synopsis: Return X (horizontal) cursor position

Declaration: `function WhereX : Byte`

Visibility: default

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 (387), WhereY (393), Window (394)

Listing: ./crtex/ex7.pp

```
Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  WriteLn('Cursor postion: X=',WhereX,' Y=',WhereY);
end.
```

3.3.22 WhereY

Synopsis: Return Y (vertical) cursor position

Declaration: `function WhereY : Byte`

Visibility: default

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 (387), WhereX (393), Window (394)

Listing: ./crtex/ex7.pp

```
Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  WriteLn('Cursor postion: X=',WhereX,' Y=',WhereY);
end.
```

3.3.23 Window

Synopsis: Create new window on screen.

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

Visibility: default

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 (387), WhereX (393), WhereY (393), ClrScr (384)

Listing: ./crtex/ex5.pp

```
Program Example5;
uses Crt;

{ Program to demonstrate the Window function. }

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

Chapter 4

Reference for unit 'dateutils'

4.1 Used units

Table 4.1: Used units by unit 'dateutils'

Name	Page
math	684
sysutils	1350
Types	395

4.2 Overview

DateUtils contains a large number of date/time manipulation routines, all based on the `TDateTime` type. There are routines for date/time math, for comparing dates and times, for composing dates and decomposing dates in their constituent parts.

4.3 Constants, types and variables

4.3.1 Constants

```
ApproxDaysPerMonth : Double = 30.4375
```

Average number of days in a month, measured over a year. Used in MonthsBetween (441).

```
ApproxDaysPerYear : Double = 365.25
```

Average number of days in a year, measured over 4 years. Used in YearsBetween (481).

```
DayFriday = 5
```

ISO day number for Friday

```
DayMonday = 1
```

ISO day number for Monday

```
DaySaturday = 6
```

ISO day number for Saturday

```
DaysPerWeek = 7
```

Number of days in a week.

```
DaysPerYear : Array[Boolean] of Word = (365,366 )
```

Array with number of days in a year. The boolean index indicates whether it is a leap year or not.

```
DaySunday = 7
```

ISO day number for Sunday

```
DayThursday = 4
```

ISO day number for Thursday

```
DayTuesday = 2
```

ISO day number for Tuesday

```
DayWednesday = 3
```

ISO day number for Wednesday

```
MonthsPerYear = 12
```

Number of months in a year

```
OneHour = 1 / HoursPerDay
```

One hour as a fraction of a day (suitable for TDateTime)

```
OneMillisecond = 1 / MSecsPerDay
```

One millisecond as a fraction of a day (suitable for TDateTime)

```
OneMinute = 1 / MinsPerDay
```

One minute as a fraction of a day (suitable for TDateTime)

```
OneSecond = 1 / SecsPerDay
```

One second as a fraction of a day (suitable for TDateTime)

```
RecodeLeaveFieldAsIs = High ( Word )
```

Bitmask deciding what to do with each TDateTime field in recode routines

`WeeksPerFortnight = 2`

Number of weeks in fortnight

`YearsPerCentury = 100`

Number of years in a century

`YearsPerDecade = 10`

Number of years in a decade

`YearsPerMillennium = 1000`

Number of years in a millenium

4.4 Procedures and functions

4.4.1 CompareDate

Synopsis: Compare 2 dates, disregarding the time of day

Declaration:
```
function CompareDate(const A: TDateTime;const B: TDateTime)
                          : TValueRelationship
```

Visibility: default

Description: `CompareDate` compares the date parts of two timestamps `A` and `B` and returns the following results:

< 0 if the day part of `A` is earlier than the day part of `B`.

0 if `A` and `B` are the on same day (times may differ) .

> 0 if the day part of `A` is later than the day part of `B`.

See also: CompareTime (399), CompareDateTime (398), SameDate (450), SameTime (452), SameDateTime (451)

Listing: ./datutex/ex99.pp

```
Program Example99;

{ This program demonstrates the CompareDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy';

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;
```

```
begin
  Write(FormatDateTime(Fmt,D1),' is ');
  Cmp:=CompareDate(D1,D2);
  If Cmp<0 then
    write('earlier than ')
  else if Cmp>0 then
    Write('later than ')
  else
    Write('equal to ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(D+1,D);
  Test(D-1,D);
  Test(D+OneSecond,D);
  Test(D-OneSecond,D);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

4.4.2 CompareDateTime

Synopsis: Compare 2 dates, taking into account the time of day

Declaration:
```
function CompareDateTime(const A: TDateTime;const B: TDateTime)
                                : TValueRelationship
```

Visibility: default

Description: `CompareDateTime` compares two timestamps `A` and `B` and returns the following results:

< 0 if `A` is earlier in date/time than `B`.

0 if `A` and `B` are the same date/time .

> 0 if `A` is later in date/time than `B`.

See also: CompareTime (399), CompareDate (397), SameDate (450), SameTime (452), SameDateTime (451)

Listing: ./datutex/ex98.pp

```
Program Example98;

{ This program demonstrates the CompareDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';
```

```
Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;

begin
  Write(FormatDateTime(Fmt,D1),' is ');
  Cmp:=CompareDateTime(D1,D2);
  If Cmp<0 then
    write('earlier than ')
  else if Cmp>0 then
    Write('later than ')
  else
    Write('equal to ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(D+1,D);
  Test(D-1,D);
  Test(D+OneSecond,D);
  Test(D-OneSecond,D);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

4.4.3 CompareTime

Synopsis: Compares two times of the day, disregarding the date part.

Declaration:
```
function CompareTime(const A: TDateTime;const B: TDateTime)
                          : TValueRelationship
```

Visibility: default

Description: `CompareTime` compares the time parts of two timestamps `A` and `B` and returns the following results:

< 0 if the time part of `A` is earlier than the time part of `B`.

0 if `A` and `B` have the same time part (dates may differ) .

> 0 if the time part of `A` is later than the time part of `B`.

See also: CompareDateTime (398), CompareDate (397), SameDate (450), SameTime (452), SameDateTime (451)

Listing: ./datutex/ex100.pp

```
Program Example100;

{ This program demonstrates the CompareTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;

begin
  Write(FormatDateTime(Fmt,D1),' has ');
  Cmp:=CompareDateTime(D1,D2);
  If Cmp<0 then
    write('earlier time than ')
  else if Cmp>0 then
    Write('later time than ')
  else
    Write('equal time with ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

4.4.4 DateOf

Synopsis: Extract the date part from a DateTime indication.

Declaration: `function DateOf(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `DateOf` extracts the date part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the date part encoded in the integer part, this operation corresponds to a call to `Trunc`.

See also: TimeOf (462), YearOf (481), MonthOf (440), DayOf (402), HourOf (417), MinuteOf (436), SecondOf (453), MilliSecondOf (432)

Listing: ./datutex/ex1.pp

```
Program Example1;

{ This program demonstrates the DateOf function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('Date is: ',DateTimeToStr(DateOf(Now)));
End.
```

4.4.5 DateTimeToJulianDate

Synopsis: Converts a TDateTime value to a Julian date representation

Declaration: `function DateTimeToJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: JulianDateToDateTime (431), TryJulianDateToDateTime (467), DateTimeToModifiedJulianDate (401), TryModifiedJulianDateToDateTime (467)

4.4.6 DateTimeToMac

Synopsis: Convert a `TDateTime` timestamp to a Mac timestamp

Declaration: `function DateTimeToMac(const AValue: TDateTime) : Int64`

Visibility: default

Description: `DateTimeToMac` converts the `TDateTime` value `AValue` to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: UnixTimeStampToMac (468), MacToDateTime (432), MacTimeStampToUnix (432)

4.4.7 DateTimeToModifiedJulianDate

Synopsis: Convert a `TDateTime` value to a modified Julian date representation

Declaration: `function DateTimeToModifiedJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (401), JulianDateToDateTime (431), TryJulianDateToDateTime (467), TryModifiedJulianDateToDateTime (467)

4.4.8 DateTimeToUnix

Synopsis: Convert a `TDateTime` value to Unix epoch time

Declaration: `function DateTimeToUnix(const AValue: TDateTime) : Int64`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: UnixToDateTime (468)

4.4.9 DayOf

Synopsis: Extract the day (of month) part from a DateTime value

Declaration: `function DayOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOf` returns the day of the month part of the `AValue` date/time indication. It is a number between 1 and 31.

For an example, see YearOf (481)

See also: YearOf (481), WeekOf (469), MonthOf (440), HourOf (417), MinuteOf (436), SecondOf (453), MilliSecondOf (432)

4.4.10 DayOfTheMonth

Synopsis: Extract the day (of month) part of a DateTime value

Declaration: `function DayOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheMonth` returns the number of days that have passed since the start of the month till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the month will return 1.

For an example, see the WeekOfTheMonth (469) function.

See also: DayOfTheYear (403), WeekOfTheMonth (469), HourOfTheMonth (417), MinuteOfTheMonth (437), SecondOfTheMonth (455), MilliSecondOfTheMonth (433)

4.4.11 DayOfTheWeek

Synopsis: Extracts the day of the week from a DateTime value

Declaration: `function DayOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheWeek` returns the number of days that have passed since the start of the week till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the week will return 1.

See also: DayOfTheYear (403), DayOfTheMonth (402), HourOfTheWeek (418), MinuteOfTheWeek (438), SecondOfTheWeek (455), MilliSecondOfTheWeek (434)

Listing: ./datutex/ex42.pp

```
Program Example42;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Day of the Week          : ',DayOfTheWeek(N));
  Writeln('Hour of the Week         : ',HourOfTheWeek(N));
  Writeln('Minute of the Week       : ',MinuteOfTheWeek(N));
  Writeln('Second of the Week       : ',SecondOfTheWeek(N));
  Writeln('MilliSecond of the Week : ',
          MilliSecondOfTheWeek(N));
End.
```

4.4.12 DayOfTheYear

Synopsis: Extracts the day of the year from a `TDateTime` value

Declaration: `function DayOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheYear` returns the number of days that have passed since the start of the year till the moment indicated by `AValue`. This is a one-based number, i.e. January 1 will return 1.

For an example, see the WeekOfTheYear (470) function.

See also: WeekOfTheYear (470), HourOfTheYear (418), MinuteOfTheYear (438), SecondOfTheYear (455), MilliSecondOfTheYear (434)

4.4.13 DaysBetween

Synopsis: Number of whole days between two DateTime values.

Declaration: `function DaysBetween(const ANow: TDateTime;const AThen: TDateTime) : Integer`

Visibility: default

Description: `DaysBetween` returns the number of whole days between `ANow` and `AThen`. This means the fractional part of a day (hours, minutes, etc.) is dropped.

See also: YearsBetween (481), MonthsBetween (441), WeeksBetween (470), HoursBetween (418), MinutesBetween (438), SecondsBetween (455), MilliSecondsBetween (435)

Listing: ./datutex/ex58.pp

```
Program Example58;

{ This program demonstrates the DaysBetween function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of days between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  WriteIn(' : ',DaysBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2);
  D2:=Today-1;
  Test(D1,D2);
  D2:=Today-25/24;
  Test(D1,D2);
  D2:=Today-26/24;
  Test(D1,D2);
  D2:=Today-5.4;
  Test(D1,D2);
  D2:=Today-2.5;
  Test(D1,D2);
End.
```

4.4.14 DaysInAMonth

Synopsis: Number of days in a month of a certain year.

Declaration: `function DaysInAMonth(const AYear: Word;const AMonth: Word) : Word`

Visibility: default

Description: `DaysInYMonth` returns the number of days in the month `AMonth` in the year`AYear`. The return value takes leap years into account.

See also: WeeksInAYear (471), WeeksInYear (472), DaysInYear (406), DaysInAYear (405), DaysInMonth (405)

Listing: ./datutex/ex17.pp

```
Program Example17;

{ This program demonstrates the DaysInAMonth function }

Uses SysUtils,DateUtils;

Var
  Y,M : Word;
```

```
Begin
  For Y:=1992 to 2010 do
    For M:=1 to 12 do
      WriteIn(LongMonthNames[m],' ',Y,' has ',DaysInAMonth(Y,M),' days.');
End.
```

4.4.15 DaysInAYear

Synopsis: Number of days in a particular year.

Declaration: `function DaysInAYear(const AYear: Word) : Word`

Visibility: default

Description: `DaysInAYear` returns the number of weeks in the year `AYear`. The return value is either 365 or 366.

See also: WeeksInAYear (471), WeeksInYear (472), DaysInYear (406), DaysInMonth (405), DaysInAMonth (404)

Listing: ./datutex/ex15.pp

```
Program Example15;

{ This program demonstrates the DaysInAYear function }

Uses SysUtils,DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteIn(Y,' has ',DaysInAYear(Y),' days.');
End.
```

4.4.16 DaysInMonth

Synopsis: Return the number of days in the month in which a date occurs.

Declaration: `function DaysInMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DaysInMonth` returns the number of days in the month in which `AValue` falls. The return value takes leap years into account.

See also: WeeksInAYear (471), WeeksInYear (472), DaysInYear (406), DaysInAYear (405), DaysInAMonth (404)

Listing: ./datutex/ex16.pp

```
Program Example16;

{ This program demonstrates the DaysInMonth function }
```

```
Uses SysUtils,DateUtils;

Var
  Y,M : Word;

Begin
  For Y:=1992 to 2010 do
    For M:=1 to 12 do
      WriteIn(LongMonthNames[m],' ',Y,' has ',DaysInMonth(EncodeDate(Y,M,1)),' days.');
End.
```

4.4.17 DaysInYear

Synopsis: Return the number of days in the year in which a date occurs.

Declaration: `function DaysInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `daysInYear` returns the number of days in the year part of `AValue`. The return value is either 365 or 366.

See also: WeeksInAYear (471), WeeksInYear (472), DaysInAYear (405), DaysInMonth (405), DaysInAMonth (404)

Listing: ./datutex/ex14.pp

```
Program Example14;

{ This program demonstrates the DaysInYear function }

Uses SysUtils,DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteIn(Y,' has ',DaysInYear(EncodeDate(Y,1,1)),' days.');
End.
```

4.4.18 DaySpan

Synopsis: Calculate the approximate number of days between two DateTime values.

Declaration: `function DaySpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `DaySpan` returns the number of Days between `ANow` and `AThen`, including any fractional parts of a Day.

See also: YearSpan (482), MonthSpan (442), WeekSpan (472), HourSpan (419), MinuteSpan (439), SecondSpan (456), MilliSecondSpan (435), DaysBetween (403)

Listing: ./datutex/ex66.pp

```
Program Example66;

{ This program demonstrates the DaySpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of days between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',DaySpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2);
  D2:=Today-1;
  Test(D1,D2);
  D2:=Today-25/24;
  Test(D1,D2);
  D2:=Today-26/24;
  Test(D1,D2);
  D2:=Today-5.4;
  Test(D1,D2);
  D2:=Today-2.5;
  Test(D1,D2);
End.
```

4.4.19 DecodeDateDay

Synopsis: Decode a DateTime value in year and year of day.

Declaration:
```
procedure DecodeDateDay(const AValue: TDateTime;var AYear: Word;
                        var ADayOfYear: Word)
```

Visibility: default

Description: `DecodeDateDay` decomposes the date indication in `AValue` and returns the various components in `AYear`, `ADayOfYear`.

See also: EncodeDateTime (411), EncodeDateMonthWeek (411), EncodeDateWeek (411), EncodeDateDay (410), DecodeDateTime (408), DecodeDateWeek (409), DecodeDateMonthWeek (408)

Listing: ./datutex/ex83.pp

```
Program Example83;

{ This program demonstrates the DecodeDateDay function }

Uses SysUtils, DateUtils;
```

```
Var
  Y,DoY : Word;
  TS : TDateTime;

Begin
  DecodeDateDay(Now,Y,DoY);
  TS:=EncodeDateDay(Y,DoY);
  Writeln('Today is : ',DateToStr(TS));
End.
```

4.4.20 DecodeDateMonthWeek

Synopsis: Decode a DateTime value in a month, week of month and day of week

Declaration:
```
procedure DecodeDateMonthWeek(const AValue: TDateTime;var AYear: Word;
                              var AMonth: Word;var AWeekOfMonth: Word;
                              var ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDateMonthWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AMonthAWeekOfMonth` and `ADayOfWeek`.

See also: EncodeDateTime (411), EncodeDateMonthWeek (411), EncodeDateWeek (411), EncodeDateDay (410), DecodeDateTime (408), DecodeDateWeek (409), DecodeDateDay (407)

Listing: ./datutex/ex85.pp

```
Program Example85;

{ This program demonstrates the DecodeDateMonthWeek function }

Uses SysUtils,DateUtils;

Var
  Y,M,Wom,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateMonthWeek(Now,Y,M,WoM,DoW);
  TS:=EncodeDateMonthWeek(Y,M,WoM,Dow);
  Writeln('Today is : ',DateToStr(TS));
End.
```

4.4.21 DecodeDateTime

Synopsis: Decode a datetime value in a date and time value

Declaration:
```
procedure DecodeDateTime(const AValue: TDateTime;var AYear: Word;
                         var AMonth: Word;var ADay: Word;var AHour: Word;
                         var AMinute: Word;var ASecond: Word;
                         var AMilliSecond: Word)
```

Visibility: default

Description: `DecodeDateTime` decomposes the date/time indication in `AValue` and returns the various components in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond`, `AMilliSecond`

See also: EncodeDateTime (411), EncodeDateMonthWeek (411), EncodeDateWeek (411), EncodeDateDay (410), DecodeDateWeek (409), DecodeDateDay (407), DecodeDateMonthWeek (408)

Listing: ./datutex/ex79.pp

```
Program Example79;

{ This program demonstrates the DecodeDateTime function }

Uses SysUtils, DateUtils;

Var
  Y,Mo,D,H,Mi,S,MS : Word;
  TS : TDateTime;

Begin
  DecodeDateTime(Now,Y,Mo,D,H,Mi,S,MS);
  TS:=EncodeDateTime(Y,Mo,D,H,Mi,S,MS);
  Writeln('Now is : ',DateTimeToStr(TS));
End.
```

4.4.22 DecodeDateWeek

Synopsis: Decode a DateTime value in a week of year and day of week.

Declaration:
```
procedure DecodeDateWeek(const AValue: TDateTime;var AYear: Word;
                         var AWeekOfYear: Word;var ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDateWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AWeekOfYear`, `ADayOfWeek`.

See also: EncodeDateTime (411), EncodeDateMonthWeek (411), EncodeDateWeek (411), EncodeDateDay (410), DecodeDateTime (408), DecodeDateDay (407), DecodeDateMonthWeek (408)

Listing: ./datutex/ex81.pp

```
Program Example81;

{ This program demonstrates the DecodeDateWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateWeek(Now,Y,W,Dow);
  TS:=EncodeDateWeek(Y,W,Dow);
  Writeln('Today is : ',DateToStr(TS));
End.
```

4.4.23 DecodeDayOfWeekInMonth

Synopsis: Decode a DateTime value in year, month, day of week parts

Declaration:
```
procedure DecodeDayOfWeekInMonth(const AValue: TDateTime;
                                 var AYear: Word;var AMonth: Word;
                                 var ANthDayOfWeek: Word;
                                 var ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDayOfWeekInMonth` decodes the date `AValue` in a `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek`. (This is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.)

See also: NthDayOfWeek (442), EncodeDateMonthWeek (411), #rtl.sysutils.DayOfWeek (1398), EncodeDayOfWeekInMonth (412), TryEncodeDayOfWeekInMonth (466)

Listing: ./datutex/ex105.pp

```
Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils,DateUtils;

Var
  Y,M,NDoW,DoW : Word;
  D : TDateTime;
Begin
  DecodeDayOfWeekInMonth(Date,Y,M,NDoW,DoW);
  D:=EncodeDayOfWeekInMonth(Y,M,NDoW,DoW);
  Write(DateToStr(D),' is the ',NDow,'-th ');
  WriteIn(formatdateTime('dddd',D),' of the month.');
End.
```

4.4.24 EncodeDateDay

Synopsis: Encodes a year and day of year to a DateTime value

Declaration:
```
function EncodeDateDay(const AYear: Word;const ADayOfYear: Word)
                       : TDateTime
```

Visibility: default

Description: `EncodeDateDay` encodes the values `AYear` and `ADayOfYear` to a date value and returns this value.

For an example, see DecodeDateDay (407).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: EncodeDateMonthWeek (411), DecodeDateDay (407), EncodeDateTime (411), EncodeDateWeek (411), TryEncodeDateTime (464), TryEncodeDateMonthWeek (464), TryEncodeDateWeek (465)

4.4.25 EncodeDateMonthWeek

Synopsis: Encodes a year, month, week of month and day of week to a DateTime value

Declaration:
```
function EncodeDateMonthWeek(const AYear: Word;const AMonth: Word;
                             const AWeekOfMonth: Word;
                             const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `WeekOfMonth`,`ADayOfWeek`, to a date value and returns this value.

For an example, see DecodeDateMonthWeek (408).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: DecodeDateMonthWeek (408), EncodeDateTime (411), EncodeDateWeek (411), EncodeDateDay (410), TryEncodeDateTime (464), TryEncodeDateWeek (465), TryEncodeDateMonthWeek (464), TryEncodeDateDay (463), NthDayOfWeek (442)

4.4.26 EncodeDateTime

Synopsis: Encodes a DateTime value from all its parts

Declaration:
```
function EncodeDateTime(const AYear: Word;const AMonth: Word;
                        const ADay: Word;const AHour: Word;
                        const AMinute: Word;const ASecond: Word;
                        const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`,`AHour`, `AMinute`,`ASecond` and `AMilliSecond` to a date/time valueand returns this value.

For an example, see DecodeDateTime (408).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: DecodeDateTime (408), EncodeDateMonthWeek (411), EncodeDateWeek (411), EncodeDateDay (410), TryEncodeDateTime (464), TryEncodeDateWeek (465), TryEncodeDateDay (463), TryEncodeDateMonthWeek (464)

4.4.27 EncodeDateWeek

Synopsis: Encode a `TDateTime` value from a year, week and day of week triplet

Declaration:
```
function EncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                        const ADayOfWeek: Word) : TDateTime
function EncodeDateWeek(const AYear: Word;const AWeekOfYear: Word)
                         : TDateTime
```

Visibility: default

Description: `EncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value.

For an example, see DecodeDateWeek (409).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: EncodeDateMonthWeek (411), DecodeDateWeek (409), EncodeDateTime (411), EncodeDateDay (410), TryEncodeDateTime (464), TryEncodeDateWeek (465), TryEncodeDateMonthWeek (464)

4.4.28 EncodeDayOfWeekInMonth

Synopsis: Encodes a year, month, week, day of week specification to a `TDateTime` value

Declaration:
```
function EncodeDayOfWeekInMonth(const AYear: Word;const AMonth: Word;
                                const ANthDayOfWeek: Word;
                                const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek` to a valid date stamp and returns the result.

`ANthDayOfweek`is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.

For an example, see DecodeDayOfWeekInMonth (410).

Errors: If any of the values is not in range, then an `EConvertError` exception will be raised.

See also: NthDayOfWeek (442), EncodeDateMonthWeek (411), #rtl.sysutils.DayOfWeek (1398), DecodeDayOfWeekInMonth (410), TryEncodeDayOfWeekInMonth (466)

4.4.29 EndOfADay

Synopsis: Calculates a DateTime value representing the end of a specified day

Declaration:
```
function EndOfADay(const AYear: Word;const AMonth: Word;
                   const ADay: Word) : TDateTime;  Overload
function EndOfADay(const AYear: Word;const ADayOfYear: Word) : TDateTime
                   ;  Overload
```

Visibility: default

Description: `EndOfADay` returns a `TDateTime` value with the date/time indication of the last moment (23:59:59.999) of the day given by `AYear`, `AMonth`, `ADay`.

The day may also be indicated with a `AYear`, `ADayOfYear` pair.

See also: StartOfTheDay (460), StartOfADay (457), StartOfTheWeek (461), StartOfAWeek (458), StartOfAMonth (458), StartOfTheMonth (460), EndOfTheWeek (416), EndOfAWeek (413), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413), EndOfTheDay (414)

Listing: ./datutex/ex39.pp

```
Program Example39;

{ This program demonstrates the EndOfADay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"End of the day : "dd mmmm yyyy hh:nn:ss';

Var
  Y,M,D : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
```

```
  D:=DayOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfADay(Y,M,D)));
  DecodeDateDay(Today,Y,D);
  WriteLn(FormatDateTime(Fmt,EndOfADay(Y,D)));
End.
```

4.4.30 EndOfAMonth

Synopsis: Calculate a datetime value representing the last day of the indicated month

Declaration: `function EndOfAMonth(const AYear: Word;const AMonth: Word) : TDateTime`

Visibility: default

Description: `EndOfAMonth` e returns a `TDateTime` value with the date of the last day of the month indicated by the `AYear`, `AMonth` pair.

See also: StartOfTheMonth (460), StartOfAMonth (458), EndOfTheMonth (415), EndOfTheYear (416), EndOfAYear (414), StartOfAWeek (458), StartOfTheWeek (461)

Listing: ./datutex/ex31.pp

```
Program Example31;

{ This program demonstrates the EndOfAMonth function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"Last day of this month : "dd mmmm yyyy';
Var
  Y,M : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfAMonth(Y,M)));
End.
```

4.4.31 EndOfAWeek

Synopsis: Return the last moment of day of the week, given a year and a week in the year.

Declaration:
```
function EndOfAWeek(const AYear: Word;const AWeekOfYear: Word;
                    const ADayOfWeek: Word) : TDateTime
function EndOfAWeek(const AYear: Word;const AWeekOfYear: Word)
                     : TDateTime
```

Visibility: default

Description: `EndOfAWeek` returns a `TDateTime` value with the date of the last moment (23:59:59:999) on the indicated day of the week indicated by the `AYear`, `AWeek`, `ADayOfWeek` values.

The default value for `ADayOfWeek` is 7.

See also: StartOfTheWeek (461), EndOfTheWeek (416), EndOfAWeek (413), StartOfAMonth (458), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413)

Listing: ./datutex/ex35.pp

```
Program Example35;

{ This program demonstrates the EndOfAWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Last day of this week : "dd mmmm yyyy hh:nn:ss';
  Fmt2 = '"Last-1 day of this week : "dd mmmm yyyy hh:nn:ss';

Var
  Y,W : Word;

Begin
  Y:=YearOf(Today);
  W:=WeekOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfAWeek(Y,W)));
  WriteLn(FormatDateTime(Fmt2,EndOfAWeek(Y,W,6)));
End.
```

4.4.32 EndOfAYear

Synopsis: Calculate a DateTime value representing the last day of a year

Declaration: `function EndOfAYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the last day of the year `AYear` (December 31).

See also: StartOfTheYear (461), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413), StartOfAWeek (458), StartOfTheWeek (461)

Listing: ./datutex/ex27.pp

```
Program Example27;

{ This program demonstrates the EndOfAYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Last day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt,EndOfAYear(YearOf(Today))));
End.
```

4.4.33 EndOfTheDay

Synopsis: Calculate a datetime value that represents the end of a given day.

Declaration: `function EndOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/-time indication of the last moment (23:59:59.999) of this day.

See also: StartOftheDay (460), StartOfADay (457), StartOfTheWeek (461), StartOfAWeek (458), StartOfAMonth (458), StartOfTheMonth (460), EndOfTheWeek (416), EndOfAWeek (413), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413), EndOfADay (412)

Listing: ./datutex/ex37.pp

```
Program Example37;

{ This program demonstrates the EndOfTheDay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"End of the day : "dd mmmm yyyy hh:nn:ss';


Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheDay(Today)));
End.
```

4.4.34 EndOfTheMonth

Synopsis: Calculate a DateTime value representing the last day of the month, given a day in that month.

Declaration: `function EndOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the EndOfAMonth (413) function.

See also: StartOfAMonth (458), StartOfTheMonth (460), EndOfAMonth (413), EndOfTheYear (416), EndOfAYear (414), StartOfAWeek (458), StartOfTheWeek (461)

Listing: ./datutex/ex29.pp

```
Program Example29;

{ This program demonstrates the EndOfTheMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"last day of this month : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheMonth(Today)));
End.
```

4.4.35 EndOfTheWeek

Synopsis: Calculate a DateTime value which represents the end of a week, given a date in that week.

Declaration: `function EndOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the last day of that week as the EndOfAWeek (413) function.

See also: StartOfAWeek (458), StartOfTheWeek (461), EndOfAWeek (413), StartOfAMonth (458), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413)

Listing: ./datutex/ex33.pp

```
Program Example33;

{ This program demonstrates the EndOfTheWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"last day of this week : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheWeek(Today)));
End.
```

4.4.36 EndOfTheYear

Synopsis: Calculate a DateTime value representing the last day of a year, given a date in that year.

Declaration: `function EndOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the last day of that year (December 31), as the EndOfAYear (414) function.

See also: StartOfAYear (459), StartOfTheYear (461), EndOfTheMonth (415), EndOfAMonth (413), StartOfAWeek (458), StartOfTheWeek (461), EndOfAYear (414)

Listing: ./datutex/ex25.pp

```
Program Example25;

{ This program demonstrates the EndOfTheYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Last day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheYear(Today)));
End.
```

4.4.37 HourOf

Synopsis: Extract the hour part from a DateTime value.

Declaration: `function HourOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOf` returns the hour of the day part of the `AValue` date/time indication. It is a number between 0 and 23.

For an example, see YearOf (481)

See also: YearOf (481), WeekOf (469), MonthOf (440), DayOf (402), MinuteOf (436), SecondOf (453), MilliSecondOf (432)

4.4.38 HourOfTheDay

Synopsis: Calculate the hour of a given DateTime value

Declaration: `function HourOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheDay` returns the number of hours that have passed since the start of the day till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 will return 0.

See also: HourOfTheYear (418), HourOfTheMonth (417), HourOfTheWeek (418), MinuteOfTheDay (436), SecondOfTheDay (453), MilliSecondOfTheDay (432)

Listing: ./datutex/ex43.pp

```
Program Example43;

{ This program demonstrates the HourOfTheDay function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Hour of the Day        : ',HourOfTheDay(N));
  Writeln('Minute of the Day      : ',MinuteOfTheDay(N));
  Writeln('Second of the Day      : ',SecondOfTheDay(N));
  Writeln('MilliSecond of the Day : ',
          MilliSecondOfTheDay(N));
End.
```

4.4.39 HourOfTheMonth

Synopsis: Calculate the number of hours passed since the start of the month.

Declaration: `function HourOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheMonth` returns the number of hours that have passed since the start of the month till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (469) function.

See also: WeekOfTheMonth (469), DayOfTheMonth (402), MinuteOfTheMonth (437), SecondOfTheMonth (455), MilliSecondOfTheMonth (433)

4.4.40 HourOfTheWeek

Synopsis: Calculate the number of hours elapsed since the start of the week.

Declaration:
```
function HourOfTheWeek(const AValue: TDateTime) : Word
```

Visibility: default

Description: `HourOfTheWeek` returns the number of hours that have passed since the start of the Week till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (402) function.

See also: HourOfTheYear (418), HourOfTheMonth (417), HourOfTheDay (417), DayOfTheWeek (402), MinuteOfTheWeek (438), SecondOfTheWeek (455), MilliSecondOfTheWeek (434)

4.4.41 HourOfTheYear

Synopsis: Calculate the number of hours passed since the start of the year.

Declaration:
```
function HourOfTheYear(const AValue: TDateTime) : Word
```

Visibility: default

Description: `HourOfTheYear` returns the number of hours that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:59:59 will return 0.

For an example, see the WeekOfTheYear (470) function.

See also: WeekOfTheYear (470), DayOfTheYear (403), MinuteOfTheYear (438), SecondOfTheYear (455), MilliSecondOfTheYear (434)

4.4.42 HoursBetween

Synopsis: Calculate the number of whole hours between two DateTime values.

Declaration:
```
function HoursBetween(const ANow: TDateTime;const AThen: TDateTime)
                      : Int64
```

Visibility: default

Description: `HoursBetween` returns the number of whole hours between `ANow` and `AThen`. This means the fractional part of an hour (minutes,seconds etc.) is dropped.

See also: YearsBetween (481), MonthsBetween (441), WeeksBetween (470), DaysBetween (403), MinutesBetween (438), SecondsBetween (455), MilliSecondsBetween (435)

Listing: ./datutex/ex59.pp

```
Program Example59;

{ This program demonstrates the HoursBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of hours between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',HoursBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2);
  D2:=D1-(61*OneMinute);
  Test(D1,D2);
  D2:=D1-(122*OneMinute);
  Test(D1,D2);
  D2:=D1-(306*OneMinute);
  Test(D1,D2);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2);
End.
```

4.4.43 HourSpan

Synopsis: Calculate the approximate number of hours between two DateTime values.

Declaration: `function HourSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `HourSpan` returns the number of Hours between `ANow` and `AThen`, including any fractional parts of a Hour.

See also: YearSpan (482), MonthSpan (442), WeekSpan (472), DaySpan (406), MinuteSpan (439), SecondSpan (456), MilliSecondSpan (435), HoursBetween (418)

Listing: ./datutex/ex67.pp

```
Program Example67;

{ This program demonstrates the HourSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);
```

```
begin
  Write('Number of hours between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  WriteLn(' : ',HourSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2);
  D2:=D1-(61*OneMinute);
  Test(D1,D2);
  D2:=D1-(122*OneMinute);
  Test(D1,D2);
  D2:=D1-(306*OneMinute);
  Test(D1,D2);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2);
End.
```

4.4.44 IncDay

Synopsis: Increase a DateTime value with a number of days.

Declaration:
```
function IncDay(const AValue: TDateTime;const ANumberOfDays: Integer)
                  : TDateTime
function IncDay(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncDay` adds `ANumberOfDays` days to `AValue` and returns the resulting date/time.`ANumberOfDays` can be positive or negative.

See also: IncYear (423), #rtl.sysutils.IncMonth (1442), IncWeek (423), IncHour (421), IncMinute (422), IncSecond (422), IncMilliSecond (421)

Listing: ./datutex/ex74.pp

```
Program Example74;

{ This program demonstrates the IncDay function }

Uses SysUtils,DateUtils;

Begin
  WriteLn('One Day from today is ',DateToStr(IncDay(Today,1)));
  WriteLn('One Day ago from today is ',DateToStr(IncDay(Today,-1)));
End.
```

4.4.45 IncHour

Synopsis: Increase a DateTime value with a number of hours.

Declaration:
```
function IncHour(const AValue: TDateTime;const ANumberOfHours: Int64)
                    : TDateTime
function IncHour(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncHour` adds `ANumberOfHours` hours to `AValue` and returns the resulting date/time.`ANumberOfHours` can be positive or negative.

See also: IncYear (423), #rtl.sysutils.IncMonth (1442), IncWeek (423), IncDay (420), IncMinute (422), IncSecond (422), IncMilliSecond (421)

Listing: ./datutex/ex75.pp

```
Program Example75
;

{ This program demonstrates the IncHour function }

Uses SysUtils, DateUtils;

Begin
  Writeln('One Hour from now is ',DateTimeToStr(IncHour(Now,1)));
  Writeln('One Hour ago from now is ',DateTimeToStr(IncHour(Now,-1)));
End.
```

4.4.46 IncMilliSecond

Synopsis: Increase a DateTime value with a number of milliseconds.

Declaration:
```
function IncMilliSecond(const AValue: TDateTime;
                        const ANumberOfMilliSeconds: Int64) : TDateTime
function IncMilliSecond(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncMilliSecond` adds `ANumberOfMilliSeconds` milliseconds to `AValue` and returns the resulting date/time.`ANumberOfMilliSeconds` can be positive or negative.

See also: IncYear (423), #rtl.sysutils.IncMonth (1442), IncWeek (423), IncDay (420), IncHour (421), IncSecond (422), IncMilliSecond (421)

Listing: ./datutex/ex78.pp

```
Program Example78;

{ This program demonstrates the IncMilliSecond function }

Uses SysUtils, DateUtils;

Begin
  Writeln('One MilliSecond from now is ',TimeToStr(IncMilliSecond(Now,1)));
  Writeln('One MilliSecond ago from now is ',TimeToStr(IncMilliSecond(Now,-1)));
End.
```

4.4.47 IncMinute

Synopsis: Increase a DateTime value with a number of minutes.

Declaration:
```
function IncMinute(const AValue: TDateTime;
                   const ANumberOfMinutes: Int64) : TDateTime
function IncMinute(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncMinute` adds `ANumberOfMinutes` minutes to `AValue` and returns the resulting date/-time.`ANumberOfMinutes` can be positive or negative.

See also: IncYear (423), #rtl.sysutils.IncMonth (1442), IncWeek (423), IncDay (420), IncHour (421), IncSecond (422), IncMilliSecond (421)

Listing: ./datutex/ex76.pp

```
Program Example76;

{ This program demonstrates the IncMinute function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Minute from now is ',TimeToStr(IncMinute(Time,1)));
  WriteLn('One Minute ago from now is ',TimeToStr(IncMinute(Time,-1)));
End.
```

4.4.48 IncSecond

Synopsis: Increase a DateTime value with a number of seconds.

Declaration:
```
function IncSecond(const AValue: TDateTime;
                   const ANumberOfSeconds: Int64) : TDateTime
function IncSecond(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncSecond` adds `ANumberOfSeconds` seconds to `AValue` and returns the resulting date/-time.`ANumberOfSeconds` can be positive or negative.

See also: IncYear (423), #rtl.sysutils.IncMonth (1442), IncWeek (423), IncDay (420), IncHour (421), IncSecond (422), IncMilliSecond (421)

Listing: ./datutex/ex77.pp

```
Program Example77;

{ This program demonstrates the IncSecond function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Second from now is ',TimeToStr(IncSecond(Time,1)));
  WriteLn('One Second ago from now is ',TimeToStr(IncSecond(Time,-1)));
End.
```

4.4.49 IncWeek

Synopsis: Increase a DateTime value with a number of weeks.

Declaration:
```
function IncWeek(const AValue: TDateTime;const ANumberOfWeeks: Integer)
                  : TDateTime
function IncWeek(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncWeek` adds `ANumberOfWeeks` weeks to `AValue` and returns the resulting date/time.`ANumberOfWeeks` can be positive or negative.

See also: IncYear (423), #rtl.sysutils.IncMonth (1442), IncDay (420), IncHour (421), IncMinute (422), IncSecond (422), IncMilliSecond (421)

Listing: ./datutex/ex73.pp

```
Program Example73;

{ This program demonstrates the IncWeek function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Week from today is ',DateToStr(IncWeek(Today,1)));
  WriteLn('One Week ago from today is ',DateToStr(IncWeek(Today,-1)));
End.
```

4.4.50 IncYear

Synopsis: Increase a DateTime value with a number of years.

Declaration:
```
function IncYear(const AValue: TDateTime;const ANumberOfYears: Integer)
                  : TDateTime
function IncYear(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncYear` adds `ANumberOfYears` years to `AValue` and returns the resulting date/time. `ANumberOfYears` can be positive or negative.

See also: #rtl.sysutils.IncMonth (1442), IncWeek (423), IncDay (420), IncHour (421), IncMinute (422), IncSecond (422), IncMilliSecond (421)

Listing: ./datutex/ex71.pp

```
Program Example71;

{ This program demonstrates the IncYear function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One year from today is ',DateToStr(IncYear(Today,1)));
  WriteLn('One year ago from today is ',DateToStr(IncYear(Today,-1)));
End.
```

4.4.51 InvalidDateDayError

Synopsis: Raise an `EConvertError` exception when a day is not a valid day of a year.

Declaration:
```
procedure InvalidDateDayError(const AYear: Word;const ADayOfYear: Word)
```

Visibility: default

Description: `InvalidDateDayError` raises an EConvertError (1492) exception and formats the error message with an appropriate description made up from the parts `AYear` and `ADayOfYear`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (425), InvalidDateTimeError (424), InvalidDateMonthWeekError (424), InvalidDayOfWeekInMonthError (425)

4.4.52 InvalidDateMonthWeekError

Synopsis: Raise an `EConvertError` exception when a Year,Month,WeekOfMonth,DayofWeek is invalid.

Declaration:
```
procedure InvalidDateMonthWeekError(const AYear: Word;
                                    const AMonth: Word;
                                    const AWeekOfMonth: Word;
                                    const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateMonthWeekError` raises an EConvertError (1492) exception and formats the error message with an appropriate description made up from the parts `AYear`, `Amonth`, `AWeekOfMonth` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (425), InvalidDateTimeError (424), InvalidDateDayError (424), InvalidDayOfWeekInMonthError (425)

4.4.53 InvalidDateTimeError

Synopsis: Raise an `EConvertError` about an invalid date-time specification.

Declaration:
```
procedure InvalidDateTimeError(const AYear: Word;const AMonth: Word;
                               const ADay: Word;const AHour: Word;
                               const AMinute: Word;const ASecond: Word;
                               const AMilliSecond: Word;
                               const ABaseDate: TDateTime)
procedure InvalidDateTimeError(const AYear: Word;const AMonth: Word;
                               const ADay: Word;const AHour: Word;
                               const AMinute: Word;const ASecond: Word;
                               const AMilliSecond: Word)
```

Visibility: default

Description: `InvalidDateTimeError` raises an EConvertError (1492) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `ADay`,`AHour`, `AMinute`, `ASecond` and `AMilliSecond`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (425), InvalidDateDayError (424), InvalidDateMonthWeekError (424), InvalidDayOfWeekInMonthError (425)

4.4.54 InvalidDateWeekError

Synopsis: Raise an EConvertError with an invalid Year, WeekOfyear and DayOfWeek specification

Declaration:
```
procedure InvalidDateWeekError(const AYear: Word;
                               const AWeekOfYear: Word;
                               const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateWeekError` raises an EConvertError (1492) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AWeek`, `ADayOfWeek`

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateTimeError (424), InvalidDateDayError (424), InvalidDateMonthWeekError (424), InvalidDayOfWeekInMonthError (425)

4.4.55 InvalidDayOfWeekInMonthError

Synopsis: Raise an `EConvertError` exception when a Year,Month,NthDayofWeek,DayofWeek is invalid.

Declaration:
```
procedure InvalidDayOfWeekInMonthError(const AYear: Word;
                                       const AMonth: Word;
                                       const ANthDayOfWeek: Word;
                                       const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDayOfWeekInMonthError` raises an EConvertError (1492) exception and formats the error message with an appropriate description made up from the parts `AYear`, `Amonth`, `ANthDayOfWeek` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (425), InvalidDateTimeError (424), InvalidDateDayError (424), InvalidDateMonthWeekError (424)

4.4.56 IsInLeapYear

Synopsis: Determine whether a date is in a leap year.

Declaration:
```
function IsInLeapYear(const AValue: TDateTime) : Boolean
```

Visibility: default

Description: `IsInLeapYear` returns `True` if the year part of `AValue` is leap year, or `False` if not.

See also: YearOf (481), IsPM (426), IsToday (427), IsSameDay (426)

Listing: ./datutex/ex3.pp

```
Program Example3;

{ This program demonstrates the IsInLeapYear function }

Uses SysUtils, DateUtils;

Begin
  WriteIn('Current year is leap year: ',IsInLeapYear(Date));
End.
```

4.4.57 IsPM

Synopsis: Determine whether a time is PM or AM.

Declaration: `function IsPM(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsPM` returns `True` if the time part of `AValue` is later then 12:00 (PM, or afternoon).

See also: YearOf (481), IsInLeapYear (425), IsToday (427), IsSameDay (426)

Listing: ./datutex/ex4.pp

```
Program Example4;

{ This program demonstrates the IsPM function }

Uses SysUtils, DateUtils;

Begin
  WriteIn('Current time is PM : ',IsPM(Now));
End.
```

4.4.58 IsSameDay

Synopsis: Check if two date/time indications are the same day.

Declaration: `function IsSameDay(const AValue: TDateTime;const ABasis: TDateTime)
: Boolean`

Visibility: default

Description: `IsSameDay` checks whether `AValue` and `ABasis` have the same date part, and returns `True` if they do, `False` if not.

See also: Today (462), Yesterday (483), Tomorrow (463), IsToday (427)

Listing: ./datutex/ex21.pp

```
Program Example21;

{ This program demonstrates the IsSameDay function }

Uses SysUtils, DateUtils;
```

```
Var
  I : Integer;
  D : TDateTime;

Begin
  For I:=1 to 3 do
    begin
    D:=Today+Random(3)-1;
    Write(FormatDateTime('dd mmmm yyyy "is today : "',D));
    WriteIn(IsSameDay(D,Today));
    end;
End.
```

4.4.59 IsToday

Synopsis: Check whether a given date is today.

Declaration: `function IsToday(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsToday` returns `True` if `AValue` is today's date, and `False` otherwise.

See also: Today (462), Yesterday (483), Tomorrow (463), IsSameDay (426)

Listing: ./datutex/ex20.pp

```
Program Example20;

{ This program demonstrates the IsToday function }

Uses SysUtils,DateUtils;

Begin
  WriteIn('Today     : ',IsToday(Today));
  WriteIn('Tomorrow  : ',IsToday(Tomorrow));
  WriteIn('Yesterday : ',IsToday(Yesterday));
End.
```

4.4.60 IsValidDate

Synopsis: Check whether a set of values is a valid date indication.

Declaration: `function IsValidDate(const AYear: Word;const AMonth: Word;`
`const ADay: Word) : Boolean`

Visibility: default

Description: `IsValidDate` returns `True` when the values `AYear`, `AMonth`, `ADay` form a valid date indication. If one of the values is not valid (e.g. the day is invalid or does not exist in that particular month), `False` is returned.

`AYear` must be in the range 1..9999 to be valid.

See also: IsValidTime (431), IsValidDateTime (429), IsValidDateDay (428), IsValidDateWeek (430), IsValidDateMonthWeek (429)

Listing: ./datutex/ex5.pp

```
Program Example5;

{ This program demonstrates the IsValidDate function }

Uses SysUtils, DateUtils;

Var
  Y,M,D : Word;

Begin
  For Y:=2000 to 2004 do
   For M:=1 to 12 do
     For D:=1 to 31 do
       If Not IsValidDate(Y,M,D) then
         WriteIn(D,' is not a valid day in ',Y,'/',M);
End.
```

4.4.61 IsValidDateDay

Synopsis: Check whether a given year/day of year combination is a valid date.

Declaration:
```
function IsValidDateDay(const AYear: Word;const ADayOfYear: Word)
                        : Boolean
```

Visibility: default

Description: `IsValidDateDay` returns `True` if `AYear` and `ADayOfYear` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfYear` value is checked to see whether it falls within the valid range of dates for `AYear`.

See also: IsValidDate (427), IsValidTime (431), IsValidDateTime (429), IsValidDateWeek (430), IsValidDateMonthWeek (429)

Listing: ./datutex/ex9.pp

```
Program Example9;

{ This program demonstrates the IsValidDateDay function }

Uses SysUtils, DateUtils;

Var
  Y : Word;

Begin
  For Y:=1996 to 2004 do
    if IsValidDateDay(Y,366) then
      WriteIn(Y,' is a leap year');
End.
```

4.4.62 IsValidDateMonthWeek

Synopsis: Check whether a given year/month/week/day of the week combination is a valid day

Declaration:
```
function IsValidDateMonthWeek(const AYear: Word;const AMonth: Word;
                              const AWeekOfMonth: Word;
                              const ADayOfWeek: Word) : Boolean
```

Visibility: default

Description: `IsValidDateMonthWeek` returns `True` if `AYear`, `AMonthAWeekOfMonth` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `AWeekOfMonth`,`ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for the `AYear`,`AMonth` combination.

See also: IsValidDate (427), IsValidTime (431), IsValidDateTime (429), IsValidDateDay (428), IsValidDateWeek (430)

Listing: ./datutex/ex11.pp

```
Program Example11;

{ This program demonstrates the IsValidDateMonthWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,D : Word;
  B : Boolean;

Begin
  For Y:=2000 to 2004 do
    begin
    B:=True;
    For W:=4 to 6 do
      For D:=1 to 7 do
        If B then
          begin
          B:=IsValidDateMonthWeek(Y,12,W,D);
          If Not B then
            if (D=1) then
              Writeln('December ',Y,' has exactly ',W,' weeks.')
            else
              Writeln('December ',Y,' has ',W,' weeks and ',D-1,' days.');
          end;
    end;
End.
```

4.4.63 IsValidDateTime

Synopsis: Check whether a set of values is a valid date and time indication.

Declaration:
```
function IsValidDateTime(const AYear: Word;const AMonth: Word;
                         const ADay: Word;const AHour: Word;
                         const AMinute: Word;const ASecond: Word;
                         const AMilliSecond: Word) : Boolean
```

Visibility: default

Description: `IsValidTime` returns `True` when the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond` form a valid date and time indication. If one of the values is not valid (e.g. the seconds are larger than 60), `False` is returned.

`AYear` must be in the range 1..9999 to be valid.

See also: IsValidDate (427), IsValidTime (431), IsValidDateDay (428), IsValidDateWeek (430), IsValidDateMonthWeek (429)

Listing: ./datutex/ex7.pp

```
Program Example7;

{ This program demonstrates the IsValidDateTime function }

Uses SysUtils, DateUtils;

Var
  Y,Mo,D : Word;
  H,M,S,MS : Word;
  I : Integer;

Begin
  For I:=1 to 10 do
    begin
    Y:=2000+Random(5);
    Mo:=Random(15);
    D:=Random(40);
    H:=Random(32);
    M:=Random(90);
    S:=Random(90);
    MS:=Random(1500);
    If Not IsValidDateTime(Y,Mo,D,H,M,S,MS) then
      WriteLn(Y,'-',Mo,'-',D,' ',H,':',M,':',S,'.',MS,' is not a valid date/time.');
    end;
End.
```

4.4.64 IsValidDateWeek

Synopsis: Check whether a given year/week/day of the week combination is a valid day.

Declaration:
```
function IsValidDateWeek(const AYear: Word;const AWeekOfYear: Word;
                         const ADayOfWeek: Word) : Boolean
```

Visibility: default

Description: `IsValidDateWeek` returns `True` if `AYear`, `AWeekOfYear` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfWeek`,`ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for `AYear`.

See also: IsValidDate (427), IsValidTime (431), IsValidDateTime (429), IsValidDateDay (428), IsValidDateMonthWeek (429)

Listing: ./datutex/ex10.pp

```
Program Example10;

{ This program demonstrates the IsValidDateWeek function }

Uses SysUtils,DateUtils;

Var
  Y,W,D : Word;
  B : Boolean;

Begin
  For Y:=2000 to 2004 do
    begin
    B:=True;
    For W:=51 to 54 do
      For D:=1 to 7 do
        If B then
          begin
          B:=IsValidDateWeek(Y,W,D);
          If Not B then
            if (D=1) then
              Writeln(Y,' has exactly ',W,' weeks.')
            else
              Writeln(Y,' has ',W,' weeks and ',D-1,' days.');
          end;
    end;
End.
```

4.4.65 IsValidTime

Synopsis: Check whether a set of values is a valid time indication.

Declaration:
```
function IsValidTime(const AHour: Word;const AMinute: Word;
                     const ASecond: Word;const AMilliSecond: Word)
                      : Boolean
```

Visibility: default

Description: Check whether a set of values is a valid time indication.

4.4.66 JulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration:
```
function JulianDateToDateTime(const AValue: Double) : TDateTime
```

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (401), TryJulianDateToDateTime (467), DateTimeToModifiedJulianDate (401), TryModifiedJulianDateToDateTime (467)

4.4.67 MacTimeStampToUnix

Synopsis: Convert a Mac timestamp to a Unix timestamp

Declaration: `function MacTimeStampToUnix(const AValue: Int64) : Int64`

Visibility: default

Description: `MacTimeStampToUnix` converts the Mac timestamp indication in `AValue` to a unix timestamp indication (epoch time)

Errors: None.

See also: UnixTimeStampToMac (468), DateTimeToMac (401), MacToDateTime (432)

4.4.68 MacToDateTime

Synopsis: Convert a Mac timestamp to a `TDateTime` timestamp

Declaration: `function MacToDateTime(const AValue: Int64) : TDateTime`

Visibility: default

Description: `MacToDateTime` converts the Mac timestamp indication in `AValue` to a valid `TDateTime` indication.

Errors: None.

See also: UnixTimeStampToMac (468), DateTimeToMac (401), MacTimeStampToUnix (432)

4.4.69 MilliSecondOf

Synopsis: Extract the millisecond part from a DateTime value.

Declaration: `function MilliSecondOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MillisecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 999.

For an example, see YearOf (481)

See also: YearOf (481), WeekOf (469), MonthOf (440), DayOf (402), HourOf (417), MinuteOf (436), MilliSecondOf (432)

4.4.70 MilliSecondOfTheDay

Synopsis: Calculate the number of milliseconds elapsed since the start of the day

Declaration: `function MilliSecondOfTheDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheDay` returns the number of milliseconds that have passed since the start of the Day (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 will return 0.

For an example, see the HourOfTheDay (417) function.

See also: MilliSecondOfTheYear (434), MilliSecondOfTheMonth (433), MilliSecondOfTheWeek (434), MilliSecondOfTheHour (433), MilliSecondOfTheMinute (433), MilliSecondOfTheSecond (433), HourOfTheDay (417), MinuteOfTheDay (436), SecondOfTheDay (453)

4.4.71 MilliSecondOfTheHour

Synopsis: Calculate the number of milliseconds elapsed since the start of the hour

Declaration: `function MilliSecondOfTheHour(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheHour` returns the number of milliseconds that have passed since the start of the Hour (HH:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.000 will return 0.

For an example, see the MinuteOfTheHour (437) function.

See also: MilliSecondOfTheYear (434), MilliSecondOfTheMonth (433), MilliSecondOfTheWeek (434), MilliSecondOfTheDay (432), MilliSecondOfTheMinute (433), MilliSecondOfTheSecond (433), MinuteOfTheHour (437), SecondOfTheHour (454)

4.4.72 MilliSecondOfTheMinute

Synopsis: Calculate the number of milliseconds elapsed since the start of the minute

Declaration: `function MilliSecondOfTheMinute(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMinute` returns the number of milliseconds that have passed since the start of the Minute (HH:MM:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.000 will return 0.

For an example, see the SecondOfTheMinute (454) function.

See also: MilliSecondOfTheYear (434), MilliSecondOfTheMonth (433), MilliSecondOfTheWeek (434), MilliSecondOfTheDay (432), MilliSecondOfTheHour (433), MilliSecondOfTheMinute (433), MilliSecondOfTheSecond (433), SecondOfTheMinute (454)

4.4.73 MilliSecondOfTheMonth

Synopsis: Calculate number of milliseconds elapsed since the start of the month.

Declaration: `function MilliSecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMonth` returns the number of milliseconds that have passed since the start of the month (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the month will return 0.

For an example, see the WeekOfTheMonth (469) function.

See also: WeekOfTheMonth (469), DayOfTheMonth (402), HourOfTheMonth (417), MinuteOfTheMonth (437), SecondOfTheMonth (455), MilliSecondOfTheMonth (433)

4.4.74 MilliSecondOfTheSecond

Synopsis: Calculate the number of milliseconds elapsed since the start of the second

Declaration: `function MilliSecondOfTheSecond(const AValue: TDateTime) : Word`

Visibility: default

Description: `MilliSecondOfTheSecond` returns the number of milliseconds that have passed since the start of the second (HH:MM:SS.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:SS.000 will return 0.

See also: MilliSecondOfTheYear (434), MilliSecondOfTheMonth (433), MilliSecondOfTheWeek (434), MilliSecondOfTheDay (432), MilliSecondOfTheHour (433), MilliSecondOfTheMinute (433), SecondOfTheMinute (454)

Listing: ./datutex/ex46.pp

```
Program Example46;

{ This program demonstrates the MilliSecondOfTheSecond function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('MilliSecond of the Second : ',
          MilliSecondOfTheSecond(N));
End.
```

4.4.75 MilliSecondOfTheWeek

Synopsis: Calculate the number of milliseconds elapsed since the start of the week

Declaration: `function MilliSecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheWeek` returns the number of milliseconds that have passed since the start of the Week (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the Week will return 0.

For an example, see the DayOfTheWeek (402) function.

See also: MilliSecondOfTheYear (434), MilliSecondOfTheMonth (433), MilliSecondOfTheDay (432), MilliSecondOfTheHour (433), MilliSecondOfTheMinute (433), MilliSecondOfTheSecond (433), DayOfTheWeek (402), HourOfTheWeek (418), MinuteOfTheWeek (438), SecondOfTheWeek (455)

4.4.76 MilliSecondOfTheYear

Synopsis: Calculate the number of milliseconds elapsed since the start of the year.

Declaration: `function MilliSecondOfTheYear(const AValue: TDateTime) : Int64`

Visibility: default

Description: `MilliSecondOfTheYear` returns the number of milliseconds that have passed since the start of the year (January 1, 00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.000 will return 0.

For an example, see the WeekOfTheYear (470) function.

See also: WeekOfTheYear (470), DayOfTheYear (403), HourOfTheYear (418), MinuteOfTheYear (438), SecondOfTheYear (455), MilliSecondOfTheYear (434)

4.4.77 MilliSecondsBetween

Synopsis: Calculate the number of whole milliseconds between two DateTime values.

Declaration:
```
function MilliSecondsBetween(const ANow: TDateTime;
                             const AThen: TDateTime) : Int64
```

Visibility: default

Description: `MillisSecondsBetween` returns the number of whole milliseconds between `ANow` and `AThen`. This means a fractional part of a millisecond is dropped.

See also: YearsBetween (481), MonthsBetween (441), WeeksBetween (470), DaysBetween (403), HoursBetween (418), MinutesBetween (438), SecondsBetween (455)

Listing: ./datutex/ex62.pp

```
Program Example62;

{ This program demonstrates the MilliSecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of milliseconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  WriteIn(' : ',MilliSecondsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2);
End.
```

4.4.78 MilliSecondSpan

Synopsis: Calculate the approximate number of milliseconds between two DateTime values.

Declaration:
```
function MilliSecondSpan(const ANow: TDateTime;const AThen: TDateTime)
                         : Double
```

Visibility: default

Description: `MilliSecondSpan` returns the number of milliseconds between `ANow` and `AThen`. Since millisecond is the smallest fraction of a `TDateTime` indication, the returned number will always be an integer value.

See also: YearSpan (482), MonthSpan (442), WeekSpan (472), DaySpan (406), HourSpan (419), MinuteSpan (439), SecondSpan (456), MilliSecondsBetween (435)

Listing: ./datutex/ex70.pp

```
Program Example70;

{ This program demonstrates the MilliSecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of milliseconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  WriteIn(' : ',MilliSecondSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2);
End.
```

4.4.79 MinuteOf

Synopsis: Extract the minute part from a DateTime value.

Declaration: `function MinuteOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOf` returns the minute of the hour part of the `AValue` date/time indication. It is a number between 0 and 59.

For an example, see YearOf (481)

See also: YearOf (481), WeekOf (469), MonthOf (440), DayOf (402), HourOf (417), SecondOf (453), MilliSecondOf (432)

4.4.80 MinuteOfTheDay

Synopsis: Calculate the number of minutes elapsed since the start of the day

Declaration: `function MinuteOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheDay` returns the number of minutes that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 will return 0.

For an example, see the HourOfTheDay (417) function.

See also: MinuteOfTheYear (438), MinuteOfTheMonth (437), MinuteOfTheWeek (438), MinuteOfTheHour (437), HourOfTheDay (417), SecondOfTheDay (453), MilliSecondOfTheDay (432)

4.4.81 MinuteOfTheHour

Synopsis: Calculate the number of minutes elapsed since the start of the hour

Declaration: `function MinuteOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheHour` returns the number of minutes that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:59 will return 0.

See also: MinuteOfTheYear (438), MinuteOfTheMonth (437), MinuteOfTheWeek (438), MinuteOfTheDay (436), SecondOfTheHour (454), MilliSecondOfTheHour (433)

Listing: ./datutex/ex44.pp

```
Program Example44;

{ This program demonstrates the MinuteOfTheHour function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Minute of the Hour       : ',MinuteOfTheHour(N));
  Writeln('Second of the Hour       : ',SecondOfTheHour(N));
  Writeln('MilliSecond of the Hour : ',
          MilliSecondOfTheHour(N));
End.
```

4.4.82 MinuteOfTheMonth

Synopsis: Calculate number of minutes elapsed since the start of the month.

Declaration: `function MinuteOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheMonth` returns the number of minutes that have passed since the start of the Month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (469) function.

See also: WeekOfTheMonth (469), DayOfTheMonth (402), HourOfTheMonth (417), MinuteOfTheMonth (437), SecondOfTheMonth (455), MilliSecondOfTheMonth (433)

4.4.83 MinuteOfTheWeek

Synopsis: Calculate the number of minutes elapsed since the start of the week

Declaration: `function MinuteOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheWeek` returns the number of minutes that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (402) function.

See also: MinuteOfTheYear (438), MinuteOfTheMonth (437), MinuteOfTheDay (436), MinuteOfTheHour (437), DayOfTheWeek (402), HourOfTheWeek (418), SecondOfTheWeek (455), MilliSecondOfTheWeek (434)

4.4.84 MinuteOfTheYear

Synopsis: Calculate the number of minutes elapsed since the start of the year

Declaration: `function MinuteOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MinuteOfTheYear` returns the number of minutes that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:59 will return 0.

For an example, see the WeekOfTheYear (470) function.

See also: WeekOfTheYear (470), DayOfTheYear (403), HourOfTheYear (418), MinuteOfTheYear (438), SecondOfTheYear (455), MilliSecondOfTheYear (434)

4.4.85 MinutesBetween

Synopsis: Calculate the number of whole minutes between two DateTime values.

Declaration:
```
function MinutesBetween(const ANow: TDateTime;const AThen: TDateTime)
                        : Int64
```

Visibility: default

Description: `MinutesBetween` returns the number of whole minutes between `ANow` and `AThen`. This means the fractional part of a minute (seconds, milliseconds etc.) is dropped.

See also: YearsBetween (481), MonthsBetween (441), WeeksBetween (470), DaysBetween (403), HoursBetween (418), SecondsBetween (455), MilliSecondsBetween (435)

Listing: ./datutex/ex60.pp

```
Program Example60;

{ This program demonstrates the MinutesBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);
```

```
begin
  Write('Number of minutes between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MinutesBetween(ANow,AThen));
end;

Var
   D1,D2 : TDateTime;

Begin
   D1:=Now;
   D2:=D1-(59*OneSecond);
   Test(D1,D2);
   D2:=D1-(61*OneSecond);
   Test(D1,D2);
   D2:=D1-(122*OneSecond);
   Test(D1,D2);
   D2:=D1-(306*OneSecond);
   Test(D1,D2);
   D2:=D1-(5.4*OneMinute);
   Test(D1,D2);
   D2:=D1-(2.5*OneMinute);
   Test(D1,D2);
End.
```

4.4.86 MinuteSpan

Synopsis: Calculate the approximate number of minutes between two DateTime values.

Declaration:
```
function MinuteSpan(const ANow: TDateTime;const AThen: TDateTime)
                          : Double
```

Visibility: default

Description: `MinuteSpan` returns the number of minutes between `ANow` and `AThen`, including any fractional parts of a minute.

See also: YearSpan (482), MonthSpan (442), WeekSpan (472), DaySpan (406), HourSpan (419), SecondSpan (456), MilliSecondSpan (435), MinutesBetween (438)

Listing: ./datutex/ex68.pp

```
Program Example68;

{ This program demonstrates the MinuteSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of minutes between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MinuteSpan(ANow,AThen));
end;

Var
```

```
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
  Test(D1,D2);
  D2:=D1-(61*OneSecond);
  Test(D1,D2);
  D2:=D1-(122*OneSecond);
  Test(D1,D2);
  D2:=D1-(306*OneSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2);
End.
```

4.4.87 ModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration: `function ModifiedJulianDateToDateTime(const AValue: Double) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (401), JulianDateToDateTime (431), TryJulianDateToDateTime (467), DateTimeToModifiedJulianDate (401), TryModifiedJulianDateToDateTime (467)

4.4.88 MonthOf

Synopsis: Extract the month from a given date.

Declaration: `function MonthOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOf` returns the month part of the `AValue` date/time indication. It is a number between 1 and 12.

For an example, see YearOf (481)

See also: YearOf (481), DayOf (402), WeekOf (469), HourOf (417), MinuteOf (436), SecondOf (453), MilliSecondOf (432)

4.4.89 MonthOfTheYear

Synopsis: Extract the month of a DateTime indication.

Declaration: `function MonthOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOfTheYear` extracts the month part of `Avalue` and returns it. It is an alias for MonthOf (440), and is provided for completeness only, corresponding to the other `PartOfTheYear` functions.

For an example, see the WeekOfTheYear (470) function.

See also: MonthOf (440), WeekOfTheYear (470), DayOfTheYear (403), HourOfTheYear (418), MinuteOfTheYear (438), SecondOfTheYear (455), MilliSecondOfTheYear (434)

4.4.90 MonthsBetween

Synopsis: Calculate the number of whole months between two DateTime values

Declaration:
```
function MonthsBetween(const ANow: TDateTime;const AThen: TDateTime)
                       : Integer
```

Visibility: default

Description: `MonthsBetween` returns the number of whole months between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years). This means the fractional part of a month is dropped.

See also: YearsBetween (481), WeeksBetween (470), DaysBetween (403), HoursBetween (418), MinutesBetween (438), SecondsBetween (455), MilliSecondsBetween (435)

Listing: ./datutex/ex56.pp

```
Program Example56;

{ This program demonstrates the MonthsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of months between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',MonthsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

4.4.91 MonthSpan

Synopsis: Calculate the approximate number of months between two DateTime values.

Declaration:
```
function MonthSpan(const ANow: TDateTime;const AThen: TDateTime)
                         : Double
```

Visibility: default

Description: `MonthSpan` returns the number of month between `ANow` and `AThen`, including any fractional parts of a month. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years).

See also: YearSpan (482), WeekSpan (472), DaySpan (406), HourSpan (419), MinuteSpan (439), SecondSpan (456), MilliSecondSpan (435), MonthsBetween (441)

Listing: ./datutex/ex64.pp

```
Program Example64;

{ This program demonstrates the MonthSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of months between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',MonthSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

4.4.92 NthDayOfWeek

Synopsis: Calculate which occurrence of weekday in the month a given day represents

Declaration:
```
function NthDayOfWeek(const AValue: TDateTime) : Word
```

Visibility: default

Description: `NthDayOfWeek` returns the occurence of the weekday of `AValue` in the month. This is the N-th time that this weekday occurs in the month (e.g. the third saturday of the month).

See also: EncodeDateMonthWeek (411), #rtl.sysutils.DayOfWeek (1398), DecodeDayOfWeekInMonth (410), EncodeDayOfWeekInMonth (412), TryEncodeDayOfWeekInMonth (466)

Listing: ./datutex/ex104.pp

```
Program Example104;

{ This program demonstrates the NthDayOfWeek function }

Uses SysUtils, DateUtils;

Begin
  Write('Today is the ',NthDayOfWeek(Today),'-th ');
  WriteLn(formatdateTime('dddd',Today),' of the month.');
End.
```

4.4.93 PreviousDayOfWeek

Synopsis: Given a day of the week, return the previous day of the week.

Declaration: `function PreviousDayOfWeek(DayOfWeek: Word) : Word`

Visibility: default

Description: `PreviousDayOfWeek` returns the previous day of the week. If the current day is the first day of the week (1) then the last day will be returned (7).

Remark: Note that the days of the week are in ISO notation, i.e. 1-based.

See also: Yesterday (483)

Listing: ./datutex/ex22.pp

```
Program Example22;

{ This program demonstrates the PreviousDayOfWeek function }

Uses SysUtils, DateUtils;

Var
  D : Word;

Begin
  For D:=1 to 7 do
    WriteLn('Previous day of ',D,' is : ',PreviousDayOfWeek(D));
End.
```

4.4.94 RecodeDate

Synopsis: Replace date part of a `TDateTime` value with another date.

Declaration:
```
function RecodeDate(const AValue: TDateTime;const AYear: Word;
                    const AMonth: Word;const ADay: Word) : TDateTime
```

Visibility: default

Description: `RecodeDate` replaces the date part of the timestamp `AValue` with the date specified in `AYear`, `AMonth`, `ADay`. All other parts (the time part) of the date/time stamp are left untouched.

Errors: If one of the `AYear`, `AMonth`, `ADay` values is not within a valid range then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex94.pp

```
Program Example94;

{ This program demonstrates the RecodeDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeDate(Now,2001,1,1));
  Writeln('This moment on the first of the millenium : ',S);
End.
```

4.4.95 RecodeDateTime

Synopsis: Replace selected parts of a `TDateTime` value with other values

Declaration:
```
function RecodeDateTime(const AValue: TDateTime;const AYear: Word;
                        const AMonth: Word;const ADay: Word;
                        const AHour: Word;const AMinute: Word;
                        const ASecond: Word;const AMilliSecond: Word)
                         : TDateTime
```

Visibility: default

Description: `RecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the pre-defined constant RecodeLeaveFieldAsIs (397), then the corresponding part of the date/time stamp is left untouched.

Errors: If one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range (`RecodeLeaveFieldAsIs` excepted) then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), TryRecodeDateTime (467)

Listing: ./datutex/ex96.pp

```
Program Example96;

{ This program demonstrates the RecodeDateTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;
  D : TDateTime ;

Begin
  D:=Now;
  D:=RecodeDateTime(D,2000,2,RecodeLeaveFieldAsIs,0,0,0,0);
  S:=FormatDateTime(Fmt,D);
  Writeln('This moment in februari 2000 : ',S);
End.
```

4.4.96 RecodeDay

Synopsis: Replace day part of a `TDateTime` value with another day.

Declaration: `function RecodeDay(const AValue: TDateTime;const ADay: Word) : TDateTime`

Visibility: default

Description: `RecodeDay` replaces the Day part of the timestamp `AValue` with `ADay`. All other parts of the date/time stamp are left untouched.

Errors: If the `ADay` value is not within a valid range (1..12) then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex89.pp

```
Program Example89;

{ This program demonstrates the RecodeDay function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeDay(Now,1));
  Writeln('This moment on the first of the month : ',S);
End.
```

4.4.97 RecodeHour

Synopsis: Replace hours part of a `TDateTime` value with another hour.

Declaration:
```
function RecodeHour(const AValue: TDateTime;const AHour: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeHour` replaces the Hour part of the timestamp `AValue` with `AHour`. All other parts of the date/time stamp are left untouched.

Errors: If the `AHour` value is not within a valid range (0..23) then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeMinute (447), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex90.pp

```
Program Example90;

{ This program demonstrates the RecodeHour function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeHour(Now,0));
  Writeln('Today, in the first hour : ',S);
End.
```

4.4.98 RecodeMilliSecond

Synopsis: Replace milliseconds part of a `TDateTime` value with another millisecond.

Declaration:
```
function RecodeMilliSecond(const AValue: TDateTime;
                           const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `RecodeMilliSecond` replaces the millisecond part of the timestamp `AValue` with `AMilliSecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMilliSecond` value is not within a valid range (0..999) then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex93.pp

```
Program Example93;

{ This program demonstrates the RecodeMilliSecond function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeMilliSecond(Now,0));
  Writeln('This moment, milliseconds stripped : ',S);
End.
```

4.4.99 RecodeMinute

Synopsis: Replace minutse part of a `TDateTime` value with another minute.

Declaration:
```
function RecodeMinute(const AValue: TDateTime;const AMinute: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeMinute` replaces the Minute part of the timestamp `AValue` with `AMinute`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMinute` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex91.pp

```
Program Example91;

{ This program demonstrates the RecodeMinute function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeMinute(Now,0));
  Writeln('This moment in the first minute of the hour: ',S);
End.
```

4.4.100 RecodeMonth

Synopsis: Replace month part of a `TDateTime` value with another month.

Declaration:
```
function RecodeMonth(const AValue: TDateTime;const AMonth: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeMonth` replaces the Month part of the timestamp `AValue` with `AMonth`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMonth` value is not within a valid range (1..12) then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex88.pp

```
Program Example88;

{ This program demonstrates the RecodeMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt, RecodeMonth(Now, 5));
  Writeln('This moment in May : ',S);
End.
```

4.4.101 RecodeSecond

Synopsis: Replace seconds part of a `TDateTime` value with another second.

Declaration:
```
function RecodeSecond(const AValue: TDateTime;const ASecond: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeSecond` replaces the Second part of the timestamp `AValue` with `ASecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `ASecond` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex92.pp

```
Program Example92;

{ This program demonstrates the RecodeSecond function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeSecond(Now,0));
  Writeln('This moment, seconds stripped : ',S);
End.
```

4.4.102 RecodeTime

Synopsis: Replace time part of a `TDateTime` value with another time.

Declaration:
```
function RecodeTime(const AValue: TDateTime;const AHour: Word;
                    const AMinute: Word;const ASecond: Word;
                    const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `RecodeTime` replaces the time part of the timestamp `AValue` with the date specified in `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. All other parts (the date part) of the date/time stamp are left untouched.

Errors: If one of the values `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range then an `EConvertError` exception is raised.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeDateTime (444)

Listing: ./datutex/ex95.pp

```
Program Example95;

{ This program demonstrates the RecodeTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeTime(Now,8,0,0,0));
  Writeln('Today, 8 AM : ',S);
End.
```

4.4.103 RecodeYear

Synopsis: Replace year part of a `TDateTime` value with another year.

Declaration:
```
function RecodeYear(const AValue: TDateTime;const AYear: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeYear` replaces the year part of the timestamp `AValue` with `AYear`. All other parts of the date/time stamp are left untouched.

Errors: If the `AYear` value is not within a valid range (1..9999) then an `EConvertError` exception is raised.

See also: RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex87.pp

```
Program Example87;

{ This program demonstrates the RecodeYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeYear(Now,1999));
  Writeln('This moment in 1999 : ',S);
End.
```

4.4.104 SameDate

Synopsis: Check whether two `TDateTime` values have the same date part.

Declaration:
```
function SameDate(const A: TDateTime;const B: TDateTime) : Boolean
```

Visibility: default

Description: `SameDate` compares the date parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareDate (397) returns zero.

See also: CompareDateTime (398), CompareDate (397), CompareTime (399), SameDateTime (451), SameTime (452)

Listing: ./datutex/ex102.pp

```
Program Example102;

{ This program demonstrates the SameDate function }
```

```
Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same date as ');
  Writeln(FormatDateTime(Fmt,D2),' : ',SameDate(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

4.4.105 SameDateTime

Synopsis: Check whether two `TDateTime` values have the same date and time parts.

Declaration: `function SameDateTime(const A: TDateTime;const B: TDateTime) : Boolean`

Visibility: default

Description: `SameDateTime` compares the date/time parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareDateTime (398) returns zero.

See also: CompareDateTime (398), CompareDate (397), CompareTime (399), SameDate (450), SameTime (452)

Listing: ./datutex/ex101.pp

```
Program Example101;

{ This program demonstrates the SameDateTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same datetime as ');
  Writeln(FormatDateTime(Fmt,D2),' : ',SameDateTime(D1,D2));
```

```
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

4.4.106 SameTime

Synopsis: Check whether two `TDateTime` values have the same time part.

Declaration: `function SameTime(const A: TDateTime;const B: TDateTime) : Boolean`

Visibility: default

Description: `SameTime` compares the time parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareTime (399) returns zero.

See also: CompareDateTime (398), CompareDate (397), CompareTime (399), SameDateTime (451), SameDate (450)

Listing: ./datutex/ex103.pp

```
Program Example102;

{ This program demonstrates the SameTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same time as ');
  WriteIn(FormatDateTime(Fmt,D2),' : ',SameTime(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
```

```
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

4.4.107 ScanDateTime

Synopsis: Scans a string for a DateTime pattern and returns the date/time

Declaration:
```
function ScanDateTime(const Pattern: String;const s: String;
                      const fmt: TFormatSettings;startpos: Integer)
                       : tdatetime;  Overload
function ScanDateTime(const Pattern: String;const s: String;
                      startpos: Integer) : tdatetime;  Overload
```

Visibility: default

Description: `ScanDateTime` scans string `S` for the date/time pattern `Pattern`, starting at position `StartPos` (default 1). Optionally, the format settings `fmt` can be specified.

In effect, this function does the opposite of what FormatDateTime (1433) does. The `Pattern` variable must contain a valid date/time pattern: note that not all possible formatdatetime patterns can be recognized, e.g., `hn` cannot be detected properly.

Errors: In case of an error, a EConvertError (395) exception is raised.

See also: FormatDateTime (395)

4.4.108 SecondOf

Synopsis: Extract the second part from a DateTime value.

Declaration:
```
function SecondOf(const AValue: TDateTime) : Word
```

Visibility: default

Description: `SecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 59.

For an example, see YearOf (481)

See also: YearOf (481), WeekOf (469), MonthOf (440), DayOf (402), HourOf (417), MinuteOf (436), MilliSecondOf (432)

4.4.109 SecondOfTheDay

Synopsis: Calculate the number of seconds elapsed since the start of the day

Declaration:
```
function SecondOfTheDay(const AValue: TDateTime) : LongWord
```

Visibility: default

Description: `SecondOfTheDay` returns the number of seconds that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 return 0.

For an example, see the HourOfTheDay (417) function.

See also: SecondOfTheYear (455), SecondOfTheMonth (455), SecondOfTheWeek (455), SecondOfTheHour (454), SecondOfTheMinute (454), HourOfTheDay (417), MinuteOfTheDay (436), MilliSecondOfTheDay (432)

4.4.110 SecondOfTheHour

Synopsis: Calculate the number of seconds elapsed since the start of the hour

Declaration: `function SecondOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheHour` returns the number of seconds that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.999 return 0.

For an example, see the MinuteOfTheHour (437) function.

See also: SecondOfTheYear (455), SecondOfTheMonth (455), SecondOfTheWeek (455), SecondOfTheDay (453), SecondOfTheMinute (454), MinuteOfTheHour (437), MilliSecondOfTheHour (433)

4.4.111 SecondOfTheMinute

Synopsis: Calculate the number of seconds elapsed since the start of the minute

Declaration: `function SecondOfTheMinute(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheMinute` returns the number of seconds that have passed since the start of the minute (HH:MM:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.999 return 0.

See also: SecondOfTheYear (455), SecondOfTheMonth (455), SecondOfTheWeek (455), SecondOfTheDay (453), SecondOfTheHour (454), MilliSecondOfTheMinute (433)

Listing: ./datutex/ex45.pp

```
Program Example45;

{ This program demonstrates the SecondOfTheMinute function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Second of the Minute      : ',SecondOfTheMinute(N));
  Writeln('MilliSecond of the Minute : ',
          MilliSecondOfTheMinute(N));
End.
```

4.4.112 SecondOfTheMonth

Synopsis: Calculate number of seconds elapsed since the start of the month.

Declaration: `function SecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheMonth` returns the number of seconds that have passed since the start of the month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (469) function.

See also: WeekOfTheMonth (469), DayOfTheMonth (402), HourOfTheMonth (417), MinuteOfTheMonth (437), MilliSecondOfTheMonth (433)

4.4.113 SecondOfTheWeek

Synopsis: Calculate the number of seconds elapsed since the start of the week

Declaration: `function SecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheWeek` returns the number of seconds that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (402) function.

See also: SecondOfTheYear (455), SecondOfTheMonth (455), SecondOfTheDay (453), SecondOfTheHour (454), SecondOfTheMinute (454), DayOfTheWeek (402), HourOfTheWeek (418), MinuteOfTheWeek (438), MilliSecondOfTheWeek (434)

4.4.114 SecondOfTheYear

Synopsis: Calculate the number of seconds elapsed since the start of the year.

Declaration: `function SecondOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheYear` returns the number of seconds that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.999 will return 0.

For an example, see the WeekOfTheYear (470) function.

See also: WeekOfTheYear (470), DayOfTheYear (403), HourOfTheYear (418), MinuteOfTheYear (438), SecondOfTheYear (455), MilliSecondOfTheYear (434)

4.4.115 SecondsBetween

Synopsis: Calculate the number of whole seconds between two DateTime values.

Declaration:
```
function SecondsBetween(const ANow: TDateTime;const AThen: TDateTime)
                        : Int64
```

Visibility: default

Description: `SecondsBetween` returns the number of whole seconds between `ANow` and `AThen`. This means the fractional part of a second (milliseconds etc.) is dropped.

See also: YearsBetween (481), MonthsBetween (441), WeeksBetween (470), DaysBetween (403), HoursBetween (418), MinutesBetween (438), MilliSecondsBetween (435)

Listing: ./datutex/ex61.pp

```
Program Example61;

{ This program demonstrates the SecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of seconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',SecondsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2);
End.
```

4.4.116 SecondSpan

Synopsis: Calculate the approximate number of seconds between two DateTime values.

Declaration:
```
function SecondSpan(const ANow: TDateTime;const AThen: TDateTime)
                          : Double
```

Visibility: default

Description: `SecondSpan` returns the number of seconds between `ANow` and `AThen`, including any fractional parts of a second.

See also: YearSpan (482), MonthSpan (442), WeekSpan (472), DaySpan (406), HourSpan (419), MinuteSpan (439), MilliSecondSpan (435), SecondsBetween (455)

Listing: ./datutex/ex69.pp

```
Program Example69;

{ This program demonstrates the SecondSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
 Write('Number of seconds between ');
 Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
 Writeln(' : ',SecondSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2);
End.
```

4.4.117 StartOfADay

Synopsis: Return the start of a day as a DateTime value, given a day indication

Declaration:
```
function StartOfADay(const AYear: Word;const AMonth: Word;
                     const ADay: Word) : TDateTime;  Overload
function StartOfADay(const AYear: Word;const ADayOfYear: Word)
                      : TDateTime;  Overload
```

Visibility: default

Description: `StartOfADay` returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of the day given by `AYear`, `AMonth`, `ADay`.

The day may also be indicated with a `AYear`, `ADayOfYear` pair.

See also: StartOfTheDay (460), StartOfTheWeek (461), StartOfAWeek (458), StartOfAMonth (458), StartOfTheMonth (460), EndOfTheWeek (416), EndOfAWeek (413), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413), EndOfTheDay (414), EndOfADay (412)

Listing: ./datutex/ex38.pp

```
Program Example38;
```

```
{ This program demonstrates the StartOfADay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Start of the day : "dd mmmm yyyy hh:nn:ss';

Var
  Y,M,D : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  D:=DayOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfADay(Y,M,D)));
  DecodeDateDay(Today,Y,D);
  WriteLn(FormatDateTime(Fmt,StartOfADay(Y,D)));
End.
```

4.4.118 StartOfAMonth

Synopsis: Return first date of month, given a year/month pair.

Declaration: `function StartOfAMonth(const AYear: Word;const AMonth: Word) : TDateTime`

Visibility: default

Description: `StartOfAMonth` e returns a `TDateTime` value with the date of the first day of the month indicated by the `AYear`, `AMonth` pair.

See also: StartOfTheMonth (460), EndOfTheMonth (415), EndOfAMonth (413), EndOfTheYear (416), EndOfAYear (414), StartOfAWeek (458), StartOfTheWeek (461)

Listing: ./datutex/ex30.pp

```
Program Example30;

{ This program demonstrates the StartOfAMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this month : "dd mmmm yyyy';
Var
  Y,M : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfAMonth(Y,M)));
End.
```

4.4.119 StartOfAWeek

Synopsis: Return a day of the week, given a year, week and day in the week.

Declaration:
```
function StartOfAWeek(const AYear: Word;const AWeekOfYear: Word;
                      const ADayOfWeek: Word) : TDateTime
function StartOfAWeek(const AYear: Word;const AWeekOfYear: Word)
                      : TDateTime
```

Visibility: default

Description: `StartOfAWeek` returns a `TDateTime` value with the date of the indicated day of the week indicated by the `AYear`, `AWeek`, `ADayOfWeek` values.

The default value for `ADayOfWeek` is 1.

See also: StartOfTheWeek (461), EndOfTheWeek (416), EndOfAWeek (413), StartOfAMonth (458), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413)

Listing: ./datutex/ex34.pp

```
Program Example34;

{ This program demonstrates the StartOfAWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this week : "dd mmmm yyyy hh:nn:ss';
  Fmt2 = '"Second day of this week : "dd mmmm yyyy hh:nn:ss';

Var
  Y,W : Word;

Begin
  Y:=YearOf(Today);
  W:=WeekOf(Today);
  WriteLn(FormatDateTime(Fmt, StartOfAWeek(Y,W)));
  WriteLn(FormatDateTime(Fmt2, StartOfAWeek(Y,W,2)));
End.
```

4.4.120 StartOfAYear

Synopsis: Return the first day of a given year.

Declaration:
```
function StartOfAYear(const AYear: Word) : TDateTime
```

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the first day of the year `AYear` (January 1).

See also: StartOfTheYear (461), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413), StartOfAWeek (458), StartOfTheWeek (461)

Listing: ./datutex/ex26.pp

```
Program Example26;

{ This program demonstrates the StartOfAYear function }

Uses SysUtils, DateUtils;
```

```
Const
  Fmt = '"First day of this year : "dd mmmm yyyy';

Begin
  WriteIn(FormatDateTime(Fmt, StartOfAYear(YearOf(Today))));
End.
```

4.4.121 StartOfTheDay

Synopsis: Calculate the start of the day as a DateTime value, given a moment in the day.

Declaration: `function StartOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of this day.

See also: StartOfADay (457), StartOfTheWeek (461), StartOfAWeek (458), StartOfAMonth (458), StartOfTheMonth (460), EndOfTheWeek (416), EndOfAWeek (413), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413), EndOftheDay (414), EndOfADay (412)

Listing: ./datutex/ex36.pp

```
Program Example36;

{ This program demonstrates the StartOfTheDay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Start of the day : "dd mmmm yyyy hh:nn:ss';


Begin
  WriteIn(FormatDateTime(Fmt, StartOfTheDay(Today)));
End.
```

4.4.122 StartOfTheMonth

Synopsis: Calculate the first day of the month, given a date in that month.

Declaration: `function StartOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the StartOfAMonth (458) function.

See also: StartOfAMonth (458), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413), StartOfAWeek (458), StartOfTheWeek (461)

Listing: ./datutex/ex28.pp

```
Program Example28;

{ This program demonstrates the StartOfTheMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this month : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheMonth(Today)));
End.
```

4.4.123 StartOfTheWeek

Synopsis: Return the first day of the week, given a date.

Declaration: `function StartOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the first day of that week as the StartOfAWeek (458) function.

See also: StartOfAWeek (458), EndOfTheWeek (416), EndOfAWeek (413), StartOfAMonth (458), EndOfTheYear (416), EndOfAYear (414), EndOfTheMonth (415), EndOfAMonth (413)

Listing: ./datutex/ex32.pp

```
Program Example32;

{ This program demonstrates the StartOfTheWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this week : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheWeek(Today)));
End.
```

4.4.124 StartOfTheYear

Synopsis: Return the first day of the year, given a date in this year.

Declaration: `function StartOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the first day of that year (January 1), as the StartOfAYear (459) function.

See also: StartOfAYear (459), EndOfTheYear (416), EndOfAYear (414)

Listing: ./datutex/ex24.pp

```
Program Example24;

{ This program demonstrates the StartOfTheYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheYear(Today)));
End.
```

4.4.125 TimeOf

Synopsis: Extract the time part from a DateTime indication.

Declaration: `function TimeOf(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `TimeOf` extracts the time part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the time part encoded in the fractional part, this operation corresponds to a call to `Frac`.

See also: DateOf (400), YearOf (481), MonthOf (440), DayOf (402), HourOf (417), MinuteOf (436), SecondOf (453), MilliSecondOf (432)

Listing: ./datutex/ex2.pp

```
Program Example2;

{ This program demonstrates the TimeOf function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('Time is : ',TimeToStr(TimeOf(Now)));
End.
```

4.4.126 Today

Synopsis: Return the current date

Declaration: `function Today : TDateTime`

Visibility: default

Description: `Today` is an alias for the Date (1394) function in the sysutils (1350) unit.

For an example, see Yesterday (483)

See also: #rtl.sysutils.Date (1394), Yesterday (483), Tomorrow (463)

4.4.127 Tomorrow

Synopsis: Return the next day

Declaration: `function Tomorrow : TDateTime`

Visibility: default

Description: `Tomorrow` returns tomorrow's date. Tomorrow is determined from the system clock, i.e. it is Today (462) +1.

See also: Today (462), Yesterday (483)

Listing: ./datutex/ex19.pp

```
Program Example19;

{ This program demonstrates the Tomorrow function }

Uses SysUtils, DateUtils;

Begin
  WriteLn(FormatDateTime('"Today is" dd mmmm yyyy',Today));
  WriteLn(FormatDateTime('"Tomorrow will be" dd mmmm yyyy',Tomorrow));
End.
```

4.4.128 TryEncodeDateDay

Synopsis: Encode a year and day of year to a `TDateTime` value

Declaration:
```
function TryEncodeDateDay(const AYear: Word;const ADayOfYear: Word;
                          var AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateDay` encodes the values `AYear` and `ADayOfYear` to a date value and returns this value in `AValue`.

If the encoding was succcesful, `True` is returned. `False` is returned if any of the arguments is not valid.

See also: EncodeDateDay (410), EncodeDateTime (411), EncodeDateMonthWeek (411), EncodeDateWeek (411), TryEncodeDateTime (464), TryEncodeDateMonthWeek (464), TryEncodeDateWeek (465)

Listing: ./datutex/ex84.pp

```
Program Example84;

{ This program demonstrates the TryEncodeDateDay function }

Uses SysUtils, DateUtils;

Var
  Y,DoY : Word;
  TS : TDateTime;

Begin
  DecodeDateDay(Now,Y,DoY);
  If TryEncodeDateDay(Y,DoY,TS) then
```

```
    WriteIn('Today is : ',DateToStr(TS))
  else
    WriteIn('Wrong year/day of year indication');
End.
```

4.4.129 TryEncodeDateMonthWeek

Synopsis: Encode a year, month, week of month and day of week to a `TDateTime` value

Declaration:
```
function TryEncodeDateMonthWeek(const AYear: Word;const AMonth: Word;
                                const AWeekOfMonth: Word;
                                const ADayOfWeek: Word;
                                var AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateTime` encodes the values `AYearAMonth`, `WeekOfMonth`,`ADayOfWeek`, to a date value and returns this value in `AValue`.

If the encoding was succesful, `True` is returned, `False` if any of the arguments is not valid.

See also: DecodeDateMonthWeek (408), EncodeDateTime (411), EncodeDateWeek (411), EncodeDateDay (410), EncodeDateMonthWeek (411), TryEncodeDateTime (464), TryEncodeDateWeek (465), TryEncodeDateDay (463), NthDayOfWeek (442)

Listing: ./datutex/ex86.pp

```
Program Example86;

{ This program demonstrates the TryEncodeDateMonthWeek function }

Uses SysUtils,DateUtils;

Var
  Y,M,Wom,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateMonthWeek(Now,Y,M,WoM,DoW);
  If TryEncodeDateMonthWeek(Y,M,WoM,Dow,TS) then
    WriteIn('Today is : ',DateToStr(TS))
  else
    WriteIn('Invalid year/month/week/dow indication');
End.
```

4.4.130 TryEncodeDateTime

Synopsis: Encode a Year, Month, Day, Hour, minute, seconds, milliseconds tuplet to a `TDateTime` value

Declaration:
```
function TryEncodeDateTime(const AYear: Word;const AMonth: Word;
                           const ADay: Word;const AHour: Word;
                           const AMinute: Word;const ASecond: Word;
                           const AMilliSecond: Word;
                           var AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`,`AHour`, `AMinute`,`ASecond` and `AMilliSecond` to a date/time valueand returns this value in `AValue`.

If the date was encoded succesfully, `True` is returned, `False` is returned if one of the arguments is not valid.

See also: EncodeDateTime (411), EncodeDateMonthWeek (411), EncodeDateWeek (411), EncodeDateDay (410), TryEncodeDateDay (463), TryEncodeDateWeek (465), TryEncodeDateMonthWeek (464)

Listing: ./datutex/ex80.pp

```
Program Example79;

{ This program demonstrates the TryEncodeDateTime function }

Uses SysUtils, DateUtils;

Var
  Y,Mo,D,H,Mi,S,MS : Word;
  TS : TDateTime;

Begin
  DecodeDateTime(Now,Y,Mo,D,H,Mi,S,MS);
  If TryEncodeDateTime(Y,Mo,D,H,Mi,S,MS,TS) then
    Writeln('Now is : ',DateTimeToStr(TS))
  else
    Writeln('Wrong date/time indication');
End.
```

4.4.131 TryEncodeDateWeek

Synopsis: Encode a year, week and day of week triplet to a `TDateTime` value

Declaration:
```
function TryEncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                           var AValue: TDateTime;const ADayOfWeek: Word)
                            : Boolean
function TryEncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                           var AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value in `AValue`.

If the encoding was succesful, `True` is returned. `False` is returned if any of the arguments is not valid.

See also: EncodeDateMonthWeek (411), EncodeDateWeek (411), EncodeDateTime (411), EncodeDateDay (410), TryEncodeDateTime (464), TryEncodeDateMonthWeek (464), TryEncodeDateDay (463)

Listing: ./datutex/ex82.pp

```
Program Example82;

{ This program demonstrates the TryEncodeDateWeek function }

Uses SysUtils, DateUtils;
```

```
Var
  Y,W,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateWeek(Now,Y,W,Dow);
  If TryEncodeDateWeek(Y,W,TS,Dow) then
    Writeln('Today is : ',DateToStr(TS))
  else
    Writeln('Invalid date/week indication');
End.
```

4.4.132 TryEncodeDayOfWeekInMonth

Synopsis: Encode a year, month, week, day of week triplet to a `TDateTime` value

Declaration:
```
function TryEncodeDayOfWeekInMonth(const AYear: Word;const AMonth: Word;
                                   const ANthDayOfWeek: Word;
                                   const ADayOfWeek: Word;
                                   var AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek` to a valid date stamp and returns the result in `AValue`.

`ANthDayOfweek`is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.

The function returns `True` if the encoding was succesful, `False` if any of the values is not in range.

See also: NthDayOfWeek (442), EncodeDateMonthWeek (411), #rtl.sysutils.DayOfWeek (1398), DecodeDayOfWeekInMonth (410), EncodeDayOfWeekInMonth (412)

Listing: ./datutex/ex106.pp

```
Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils,DateUtils;

Var
  Y,M,NDoW,DoW : Word;
  D : TDateTime;
Begin
  DecodeDayOfWeekInMonth(Date,Y,M,NDoW,DoW);
  If TryEncodeDayOfWeekInMonth(Y,M,NDoW,DoW,D) then
    begin
    Write(DateToStr(D),' is the ',NDow,'-th ');
    Writeln(formatdateTime('dddd',D),' of the month.');
    end
  else
    Writeln('Invalid year/month/NthDayOfweek combination');
End.
```

4.4.133 TryJulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration:
```
function TryJulianDateToDateTime(const AValue: Double;
                                 var ADateTime: TDateTime) : Boolean
```

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (401), JulianDateToDateTime (431), DateTimeToModifiedJulianDate (401), TryModifiedJulianDateToDateTime (467)

4.4.134 TryModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration:
```
function TryModifiedJulianDateToDateTime(const AValue: Double;
                                         var ADateTime: TDateTime)
                                          : Boolean
```

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (401), JulianDateToDateTime (431), TryJulianDateToDateTime (467), DateTimeToModifiedJulianDate (401), ModifiedJulianDateToDateTime (440)

4.4.135 TryRecodeDateTime

Synopsis: Replace selected parts of a `TDateTime` value with other values

Declaration:
```
function TryRecodeDateTime(const AValue: TDateTime;const AYear: Word;
                           const AMonth: Word;const ADay: Word;
                           const AHour: Word;const AMinute: Word;
                           const ASecond: Word;const AMilliSecond: Word;
                           var AResult: TDateTime) : Boolean
```

Visibility: default

Description: `TryRecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the pre-defined constant RecodeLeaveFieldAsIs (397), then the corresponding part of the date/time stamp is left untouched.

The resulting date/time is returned in `AValue`.

The function returns `True` if the encoding was succesful. It returns `False` if one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range.

See also: RecodeYear (450), RecodeMonth (448), RecodeDay (445), RecodeHour (446), RecodeMinute (447), RecodeSecond (448), RecodeMilliSecond (446), RecodeDate (443), RecodeTime (449), RecodeDateTime (444)

Listing: ./datutex/ex97.pp

```
Program Example97;

{ This program demonstrates the TryRecodeDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;
  D : TDateTime ;

Begin
  If TryRecodeDateTime(Now,2000,2,RecodeLeaveFieldAsIs,0,0,0,0,D) then
    begin
    S:=FormatDateTime(Fmt,D);
    WriteIn('This moment in februari 2000 : ',S);
    end
  else
    WriteIn('This moment did/does not exist in februari 2000');
End.
```

4.4.136 UnixTimeStampToMac

Synopsis: Convert Unix Timestamp to a Mac Timestamp

Declaration: `function UnixTimeStampToMac(const AValue: Int64) : Int64`

Visibility: default

Description: `UnixTimeStampToMac` converts the unix epoch time in `AValue` to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: DateTimeToMac (401), MacToDateTime (432), MacTimeStampToUnix (432)

4.4.137 UnixToDateTime

Synopsis: Convert Unix epoch time to a `TDateTime` value

Declaration: `function UnixToDateTime(const AValue: Int64) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToUnix (402)

4.4.138 WeekOf

Synopsis: Extract week (of the year) from a given date.

Declaration: `function WeekOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeekOf` returns the week-of-the-year part of the `AValue` date/time indication. It is a number between 1 and 53.

For an example, see YearOf (481)

See also: YearOf (481), DayOf (402), MonthOf (440), HourOf (417), MinuteOf (436), SecondOf (453), MilliSecondOf (432)

4.4.139 WeekOfTheMonth

Synopsis: Extract the week of the month (and optionally month and year) from a DateTime value

Declaration:
```
function WeekOfTheMonth(const AValue: TDateTime) : Word;  Overload
function WeekOfTheMonth(const AValue: TDateTime;var AYear: Word;
                        var AMonth: Word) : Word;  Overload
```

Visibility: default

Description: `WeekOfTheMonth` extracts the week of the month from `Avalue` and returns it, and optionally returns the year and month as well (in `AYear`, `AMonth` respectively).

Remark: Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year and month may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: WeekOfTheYear (470), DayOfTheMonth (402), HourOfTheMonth (417), MinuteOfTheMonth (437), SecondOfTheMonth (455), MilliSecondOfTheMonth (433)

Listing: ./datutex/ex41.pp

```
Program Example41;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Week of the Month        : ',WeekOfTheMonth(N));
  Writeln('Day of the Month         : ',DayOfTheMonth(N));
  Writeln('Hour of the Month        : ',HourOfTheMonth(N));
  Writeln('Minute of the Month      : ',MinuteOfTheMonth(N));
  Writeln('Second of the Month      : ',SecondOfTheMonth(N));
  Writeln('MilliSecond of the Month : ',
          MilliSecondOfTheMonth(N));
End.
```

4.4.140 WeekOfTheYear

Synopsis: Extract the week of the year (and optionally year) of a DateTime indication.

Declaration:
```
function WeekOfTheYear(const AValue: TDateTime) : Word;  Overload
function WeekOfTheYear(const AValue: TDateTime;var AYear: Word) : Word
                       ;  Overload
```

Visibility: default

Description: `WeekOfTheYear` extracts the week of the year from `Avalue` and returns it, and optionally returns the year as well. It returns the same value as WeekOf (469).

Remark: Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: WeekOf (469), MonthOfTheYear (440), DayOfTheYear (403), HourOfTheYear (418), MinuteOfTheYear (438), SecondOfTheYear (455), MilliSecondOfTheYear (434)

Listing: ./datutex/ex40.pp

```
Program Example40;

{ This program demonstrates the WeekOfTheYear function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Month of the year       : ',MonthOfTheYear(N));
  Writeln('Week of the year        : ',WeekOfTheYear(N));
  Writeln('Day of the year         : ',DayOfTheYear(N));
  Writeln('Hour of the year        : ',HourOfTheYear(N));
  Writeln('Minute of the year      : ',MinuteOfTheYear(N));
  Writeln('Second of the year      : ',SecondOfTheYear(N));
  Writeln('MilliSecond of the year : ',
          MilliSecondOfTheYear(N));
End.
```

4.4.141 WeeksBetween

Synopsis: Calculate the number of whole weeks between two DateTime values

Declaration:
```
function WeeksBetween(const ANow: TDateTime;const AThen: TDateTime)
                      : Integer
```

Visibility: default

Description: `WeeksBetween` returns the number of whole weeks between `ANow` and `AThen`. This means the fractional part of a Week is dropped.

See also: YearsBetween (481), MonthsBetween (441), DaysBetween (403), HoursBetween (418), MinutesBetween (438), SecondsBetween (455), MilliSecondsBetween (435)

Listing: ./datutex/ex57.pp

```
Program Example57;

{ This program demonstrates the WeeksBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of weeks between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  WriteIn(' : ',WeeksBetween(ANow,AThen));
end;

Var
   D1,D2 : TDateTime;

Begin
   D1:=Today;
   D2:=Today-7;
   Test(D1,D2);
   D2:=Today-8;
   Test(D1,D2);
   D2:=Today-14;
   Test(D1,D2);
   D2:=Today-35;
   Test(D1,D2);
   D2:=Today-36;
   Test(D1,D2);
   D2:=Today-17;
   Test(D1,D2);
End.
```

4.4.142 WeeksInAYear

Synopsis: Return the number of weeks in a given year

Declaration: `function WeeksInAYear(const AYear: Word) : Word`

Visibility: default

Description: `WeeksInAYear` returns the number of weeks in the year `AYear`. The return value is either 52 or 53.

Remark: The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a thursday.

See also: WeeksInYear (472), DaysInYear (406), DaysInAYear (405), DaysInMonth (405), DaysInAMonth (404)

Listing: ./datutex/ex13.pp

```
Program Example13;

{ This program demonstrates the WeeksInAYear function }

Uses SysUtils, DateUtils;
```

```
Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteLn(Y,' has ',WeeksInAYear(Y),' weeks.');
End.
```

4.4.143 WeeksInYear

Synopsis: return the number of weeks in the year, given a date

Declaration: `function WeeksInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeeksInYear` returns the number of weeks in the year part of `AValue`. The return value is either 52 or 53.

Remark: The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a thursday.

See also: WeeksInAYear (471), DaysInYear (406), DaysInAYear (405), DaysInMonth (405), DaysInAMonth (404)

Listing: ./datutex/ex12.pp

```
Program Example12;

{ This program demonstrates the WeeksInYear function }

Uses SysUtils,DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteLn(Y,' has ',WeeksInYear(EncodeDate(Y,2,1)),' weeks.');
End.
```

4.4.144 WeekSpan

Synopsis: Calculate the approximate number of weeks between two DateTime values.

Declaration: `function WeekSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `WeekSpan` returns the number of weeks between `ANow` and `AThen`, including any fractional parts of a week.

See also: YearSpan (482), MonthSpan (442), DaySpan (406), HourSpan (419), MinuteSpan (439), SecondSpan (456), MilliSecondSpan (435), WeeksBetween (470)

Listing: ./datutex/ex65.pp

```
Program Example57;

{ This program demonstrates the WeekSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of weeks between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',WeekSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2);
  D2:=Today-8;
  Test(D1,D2);
  D2:=Today-14;
  Test(D1,D2);
  D2:=Today-35;
  Test(D1,D2);
  D2:=Today-36;
  Test(D1,D2);
  D2:=Today-17;
  Test(D1,D2);
End.
```

4.4.145 WithinPastDays

Synopsis: Check whether two datetimes are only a number of days apart

Declaration:
```
function WithinPastDays(const ANow: TDateTime;const AThen: TDateTime;
                        const ADays: Integer) : Boolean
```

Visibility: default

Description: `WithinPastDays` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ADays` days apart, or `False` if they are further apart.

Remark: Since this function uses the DaysBetween (403) function to calculate the difference in days, this means that fractional days do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half days apart, the result will also be `True`

See also: WithinPastYears (480), WithinPastMonths (477), WithinPastWeeks (479), WithinPastHours (474), WithinPastMinutes (476), WithinPastSeconds (478), WithinPastMilliSeconds (475)

Listing: ./datutex/ex50.pp

```
Program Example50;

{ This program demonstrates the WithinPastDays function }
```

```
Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; ADays : Integer);

begin
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Write(' are within ',ADays,' days: ');
  WriteIn(WithinPastDays(ANow,AThen,ADays));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2,1);
  D2:=Today-1;
  Test(D1,D2,1);
  D2:=Today-25/24;
  Test(D1,D2,1);
  D2:=Today-26/24;
  Test(D1,D2,5);
  D2:=Today-5.4;
  Test(D1,D2,5);
  D2:=Today-2.5;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

4.4.146 WithinPastHours

Synopsis: Check whether two datetimes are only a number of hours apart

Declaration:
```
function WithinPastHours(const ANow: TDateTime;const AThen: TDateTime;
                         const AHours: Int64) : Boolean
```

Visibility: default

Description: `WithinPastHours` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AHours` hours apart, or `False` if they are further apart.

Remark: Since this function uses the HoursBetween (418) function to calculate the difference in Hours, this means that fractional hours do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half hours apart, the result will also be `True`

See also: WithinPastYears (480), WithinPastMonths (477), WithinPastWeeks (479), WithinPastDays (473), WithinPastMinutes (476), WithinPastSeconds (478), WithinPastMilliSeconds (475)

Listing: ./datutex/ex51.pp

```
Program Example51;

{ This program demonstrates the WithinPastHours function }

Uses SysUtils, DateUtils;
```

```
Procedure Test(ANow,AThen : TDateTime; AHours : Integer);

begin
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Write(' are within ',AHours,' hours: ');
  WriteIn(WithinPastHours(ANow,AThen,AHours));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(61*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(122*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(306*OneMinute);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

4.4.147 WithinPastMilliSeconds

Synopsis: Check whether two datetimes are only a number of milliseconds apart

Declaration:
```
function WithinPastMilliSeconds(const ANow: TDateTime;
                                 const AThen: TDateTime;
                                 const AMilliSeconds: Int64) : Boolean
```

Visibility: default

Description: `WithinPastMilliSeconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMilliSeconds` milliseconds apart, or `False` if they are further apart.

Remark: Since this function uses the MilliSecondsBetween (435) function to calculate the difference in milliseconds, this means that fractional milliseconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half milliseconds apart, the result will also be `True`

See also: WithinPastYears (480), WithinPastMonths (477), WithinPastWeeks (479), WithinPastDays (473), WithinPastHours (474), WithinPastMinutes (476), WithinPastSeconds (478)

Listing: ./datutex/ex54.pp

```
Program Example54;

{ This program demonstrates the WithinPastMilliSeconds function }

Uses SysUtils, DateUtils;
```

```
Procedure Test(ANow,AThen : TDateTime; AMilliSeconds : Integer);

begin
 Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
 Write(' are within ',AMilliSeconds,' milliseconds: ');
 WriteIn(WithinPastMilliSeconds(ANow,AThen,AMilliSeconds));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

4.4.148 WithinPastMinutes

Synopsis: Check whether two datetimes are only a number of minutes apart

Declaration:
```
function WithinPastMinutes(const ANow: TDateTime;const AThen: TDateTime;
                           const AMinutes: Int64) : Boolean
```

Visibility: default

Description: `WithinPastMinutes` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMinutes` minutes apart, or `False` if they are further apart.

Remark: Since this function uses the MinutesBetween (438) function to calculate the difference in Minutes, this means that fractional minutes do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half minutes apart, the result will also be `True`

See also: WithinPastYears (480), WithinPastMonths (477), WithinPastWeeks (479), WithinPastDays (473), WithinPastHours (474), WithinPastSeconds (478), WithinPastMilliSeconds (475)

Listing: ./datutex/ex52.pp

```
Program Example52;

{ This program demonstrates the WithinPastMinutes function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMinutes : Integer);

begin
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Write(' are within ',AMinutes,' Minutes: ');
```

```
  WriteLn(WithinPastMinutes(ANow,AThen,AMinutes));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(61*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(122*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(306*OneSecond);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

4.4.149 WithinPastMonths

Synopsis: Check whether two datetimes are only a number of months apart

Declaration: `function WithinPastMonths(const ANow: TDateTime;const AThen: TDateTime; const AMonths: Integer) : Boolean`

Visibility: default

Description: `WithinPastMonths` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMonths` months apart, or `False` if they are further apart.

Remark: Since this function uses the MonthsBetween (441) function to calculate the difference in Months, this means that fractional months do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half months apart, the result will also be `True`

See also: WithinPastYears (480), WithinPastWeeks (479), WithinPastDays (473), WithinPastHours (474), WithinPastMinutes (476), WithinPastSeconds (478), WithinPastMilliSeconds (475)

Listing: ./datutex/ex48.pp

```
Program Example48;

{ This program demonstrates the WithinPastMonths function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMonths : Integer);

begin
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Write(' are within ',AMonths,' months: ');
  WriteLn(WithinPastMonths(ANow,AThen,AMonths));
end;
```

```
Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2,12);
  D2:=Today-365;
  Test(D1,D2,12);
  D2:=Today-366;
  Test(D1,D2,12);
  D2:=Today-390;
  Test(D1,D2,12);
  D2:=Today-368;
  Test(D1,D2,11);
  D2:=Today-1000;
  Test(D1,D2,31);
  Test(D1,D2,32);
  Test(D1,D2,33);
End.
```

4.4.150 WithinPastSeconds

Synopsis: Check whether two datetimes are only a number of seconds apart

Declaration: `function WithinPastSeconds(const ANow: TDateTime;const AThen: TDateTime; const ASeconds: Int64) : Boolean`

Visibility: default

Description: `WithinPastSeconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ASeconds` seconds apart, or `False` if they are further apart.

Remark: Since this function uses the SecondsBetween (455) function to calculate the difference in seconds, this means that fractional seconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half seconds apart, the result will also be `True`

See also: WithinPastYears (480), WithinPastMonths (477), WithinPastWeeks (479), WithinPastDays (473), WithinPastHours (474), WithinPastMinutes (476), WithinPastMilliSeconds (475)

Listing: ./datutex/ex53.pp

```
Program Example53;

{ This program demonstrates the WithinPastSeconds function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; ASeconds : Integer);

begin
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Write(' are within ',ASeconds,' seconds: ');
 WriteIn(WithinPastSeconds(ANow,AThen,ASeconds));
end;

Var
```

```
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

4.4.151 WithinPastWeeks

Synopsis: Check whether two datetimes are only a number of weeks apart

Declaration:
```
function WithinPastWeeks(const ANow: TDateTime;const AThen: TDateTime;
                         const AWeeks: Integer) : Boolean
```

Visibility: default

Description: `WithinPastWeeks` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AWeeks` weeks apart, or `False` if they are further apart.

Remark: Since this function uses the WeeksBetween (470) function to calculate the difference in Weeks, this means that fractional Weeks do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half weeks apart, the result will also be `True`

See also: WithinPastYears (480), WithinPastMonths (477), WithinPastDays (473), WithinPastHours (474), WithinPastMinutes (476), WithinPastSeconds (478), WithinPastMilliSeconds (475)

Listing: ./datutex/ex49.pp

```
Program Example49;

{ This program demonstrates the WithinPastWeeks function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; AWeeks : Integer);

begin
 Write(DateToStr(AThen),' and ',DateToStr(ANow));
 Write(' are within ',AWeeks,' weeks: ');
 WriteIn(WithinPastWeeks(ANow,AThen,AWeeks));
end;

Var
  D1,D2 : TDateTime;
```

```
Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2,1);
  D2:=Today-8;
  Test(D1,D2,1);
  D2:=Today-14;
  Test(D1,D2,1);
  D2:=Today-35;
  Test(D1,D2,5);
  D2:=Today-36;
  Test(D1,D2,5);
  D2:=Today-17;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

4.4.152 WithinPastYears

Synopsis: Check whether two datetimes are only a number of years apart

Declaration:
```
function WithinPastYears(const ANow: TDateTime;const AThen: TDateTime;
                         const AYears: Integer) : Boolean
```

Visibility: default

Description: `WithinPastYears` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AYears` years apart, or `False` if they are further apart.

Remark: Since this function uses the YearsBetween (481) function to calculate the difference in years, this means that fractional years do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half years apart, the result will also be `True`

See also: WithinPastMonths (477), WithinPastWeeks (479), WithinPastDays (473), WithinPastHours (474), WithinPastMinutes (476), WithinPastSeconds (478), WithinPastMilliSeconds (475)

Listing: ./datutex/ex47.pp

```
Program Example47;

{ This program demonstrates the WithinPastYears function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AYears : Integer);

begin
 Write(DateToStr(AThen),' and ',DateToStr(ANow));
 Write(' are within ',AYears,' years: ');
 WriteIn(WithinPastYears(ANow,AThen,AYears));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
```

```
  D2:=Today-364;
  Test(D1,D2,1);
  D2:=Today-365;
  Test(D1,D2,1);
  D2:=Today-366;
  Test(D1,D2,1);
  D2:=Today-390;
  Test(D1,D2,1);
  D2:=Today-368;
  Test(D1,D2,1);
  D2:=Today-1000;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

4.4.153 YearOf

Synopsis: Extract the year from a given date.

Declaration: `function YearOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `YearOf` returns the year part of the `AValue` date/time indication. It is a number between 1 and 9999.

See also: MonthOf (440), DayOf (402), WeekOf (469), HourOf (417), MinuteOf (436), SecondOf (453), MilliSecondOf (432)

Listing: ./datutex/ex23.pp

```
Program Example23;

{ This program demonstrates the YearOf function }

Uses SysUtils,DateUtils;

Var
  D : TDateTime;

Begin
  D:=Now;
  Writeln('Year        : ',YearOf(D));
  Writeln('Month       : ',MonthOf(D));
  Writeln('Day         : ',DayOf(D));
  Writeln('Week        : ',WeekOf(D));
  Writeln('Hour        : ',HourOf(D));
  Writeln('Minute      : ',MinuteOf(D));
  Writeln('Second      : ',SecondOf(D));
  Writeln('MilliSecond : ',MilliSecondOf(D));
End.
```

4.4.154 YearsBetween

Synopsis: Calculate the number of whole years between two DateTime values

Declaration: `function YearsBetween(const ANow: TDateTime;const AThen: TDateTime) : Integer`

Visibility: default

Description: `YearsBetween` returns the number of whole years between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years). This means the fractional part of a year is dropped.

See also: MonthsBetween (441), WeeksBetween (470), DaysBetween (403), HoursBetween (418), MinutesBetween (438), SecondsBetween (455), MilliSecondsBetween (435), YearSpan (482)

Listing: ./datutex/ex55.pp

```
Program Example55;

{ This program demonstrates the YearsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of years between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',YearsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

4.4.155 YearSpan

Synopsis: Calculate the approximate number of years between two DateTime values.

Declaration: `function YearSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `YearSpan` returns the number of years between `ANow` and `AThen`, including any fractional parts of a year. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years).

See also: MonthSpan (442), WeekSpan (472), DaySpan (406), HourSpan (419), MinuteSpan (439), SecondSpan (456), MilliSecondSpan (435), YearsBetween (481)

Listing: ./datutex/ex63.pp

```
Program Example63;

{ This program demonstrates the YearSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of years between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',YearSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

4.4.156 Yesterday

Synopsis: Return the previous day.

Declaration: `function Yesterday : TDateTime`

Visibility: default

Description: `Yesterday` returns yesterday's date. Yesterday is determined from the system clock, i.e. it is Today (462) -1.

See also: Today (462), Tomorrow (463)

Listing: ./datutex/ex18.pp

```
Program Example18;

{ This program demonstrates the Yesterday function }
```

```
Uses SysUtils,DateUtils;

Begin
  WriteLn(FormatDateTime('"Today is " dd mmmm yyyy',Today));
  WriteLn(FormatDateTime('"Yesterday was " dd mmmm yyyy',Yesterday));
End.
```

Chapter 5

Reference for unit 'Dos'

5.1 System information

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

Table 5.1:

Name	Description
DosVersion (494)	Get OS version
GetCBreak (500)	Get setting of control-break handling flag
GetDate (500)	Get system date
GetIntVec (503)	Get interrupt vector status
GetTime (504)	Get system time
GetVerify (505)	Get verify flag
Intr (505)	Execute an interrupt
Keep (505)	Keep process in memory and exit
MSDos (506)	Execute MS-dos function call
PackTime (506)	Pack time for file time
SetCBreak (507)	Set control-break handling flag
SetDate (507)	Set system date
SetIntVec (508)	Set interrupt vectors
SetTime (509)	Set system time
SetVerify (509)	Set verify flag
SwapVectors (509)	Swap interrupt vectors
UnPackTime (510)	Unpack file time

5.2 Process handling

Functions to handle process information and starting new processes.

5.3 Directory and disk handling

Routines to handle disk information.

Table 5.2:

Name	Description
DosExitCode (494)	Exit code of last executed program
EnvCount (495)	Return number of environment variables
EnvStr (496)	Return environment string pair
Exec (496)	Execute program
GetEnv (501)	Return specified environment string

Table 5.3:

Name	Description
AddDisk (492)	Add disk to list of disks (UNIX only)
DiskFree (492)	Return size of free disk space
DiskSize (493)	Return total disk size

5.4 File handling

Routines to handle files on disk.

Table 5.4:

Name	Description
FExpand (496)	Expand filename to full path
FindClose (497)	Close finfirst/findnext session
FindFirst (497)	Start find of file
FindNext (498)	Find next file
FSearch (498)	Search for file in a path
FSplit (499)	Split filename in parts
GetFAttr (501)	Return file attributes
GetFTime (502)	Return file time
GetLongName (503)	Convert short filename to long filename (DOS only)
GetShortName (504)	Convert long filename to short filename (DOS only)
SetFAttr (507)	Set file attributes
SetFTime (508)	Set file time

5.5 File open mode constants.

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

5.6 File attributes

The File Attribute constants are used in FindFirst (497), FindNext (498) to determine what type of special file to search for in addition to normal files. These flags are also used in the SetFAttr (507) and

Table 5.5: Possible mode constants

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

GetFAttr (501) routines to set and retrieve attributes of files. For their definitions consult fileattributes (486).

Table 5.6: Possible file attributes

Constant	Description	Value
`readonly`	Read-Only file attribute	$01
`hidden`	Hidden file attribute	$02
`sysfile`	System file attribute	$04
`volumeid`	Volumd ID file attribute	$08
`directory`	Directory file attribute	$10
`archive`	Archive file attribute	$20
`anyfile`	Match any file attribute	$3F

5.7 Used units

Table 5.7: Used units by unit 'Dos'

Name	Page
baseunix	485

5.8 Overview

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

The unit was first written for dos by Florian Klaempfl. It was ported to linux by Mark May and enhanced by Michael Van Canneyt. The Amiga version was ported by Nils Sjoholm.

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

5.9 Constants, types and variables

5.9.1 Constants

```
anyfile = $3F
```

Match any file attribute

```
archive = $20
```

Archive file attribute

```
directory = $10
```

Directory file attribute

```
fauxiliary = $0010
```

CPU auxiliary flag. Not used.

```
fcarry = $0001
```

CPU carry flag. Not used.

```
FileNameLen = 255
```

Maximum length of a filename

```
filerecnamelength = 255
```

Maximum length of FileName part in FileRec (490)

```
fmclosed = $D7B0
```

File is closed

```
fminout = $D7B3
```

File is read and write

```
fminput = $D7B1
```

File is read only

```
fmoutput = $D7B2
```

File is write only

```
foverflow = $0800
```

CPU overflow flag. Not used.

```
fparity = $0004
```

CPU parity flag. Not used.

```
fsign = $0080
```

CPU sign flag. Not used.

```
fzero = $0040
```

CPU zero flag. Not used.

```
hidden = $02
```

Hidden file attribute

```
readonly = $01
```

Read-Only file attribute

```
sysfile = $04
```

System file attribute

```
TextRecBufSize = 256
```

Size of default buffer in TextRec (491)

```
TextRecNameLength = 256
```

Maximum length of filename in TextRec (491)

```
volumeid = $08
```

Volumd ID file attribute

5.9.2 Types

```
ComStr =
```

Command-line string type

```
DateTime = packed record
  Year : Word;
  Month : Word;
  Day : Word;
  Hour : Word;
  Min : Word;
  Sec : Word;
end
```

The `DateTime` type is used in PackTime (506) and UnPackTime (510) for setting/reading file times with GetFTime (502) and SetFTime (508).

```
DirStr =
```

Full directory string type.

```
ExtStr =
```

Filename extension string type.

```
FileRec = packed record
  Handle : THandle;
  Mode : LongInt;
  RecSize : SizeInt;
  _private : Array[1..3*SizeOf(SizeInt)+5*SizeOf(pointer)] of Byte;
  UserData : Array[1..32] of Byte;
  name : Array[0..filerecnamelength] of Char;
end
```

`FileRec` is used for internal representation of typed and untyped files.

```
NameStr =
```

Fill filename string type.

```
PathStr =
```

Full File path string type.

```
Registers = packed record
end
```

Record to keep CPU registers for MSDos (506) call. Unused.

```
SearchRec = packed record
  SearchPos : TOff;
  SearchNum : LongInt;
  DirPtr : Pointer;
  SearchType : Byte;
  SearchAttr : Byte;
  Mode : Word;
  Fill : Array[1..1] of Byte;
  Attr : Byte;
  Time : LongInt;
  Size : LongInt;
  Reserved : Word;
  Name : String;
  SearchSpec : String;
  NamePos : Word;
end
```

`SearchRec` is filled by the FindFirst (497) call and can be used in subsequent FindNext (498) calls to search for files. The structure of this record depends on the platform. Only the following fields are present on all platforms:

Attr File attributes.

Time File modification time.

Size File size

Name File name (name part only, no path)

Mode File access mode (linux only)

```
TextBuf = Array[0..TextRecBufSize-1] of Char
```

Type for default buffer in TextRec (491)

```
TextRec = packed record
  Handle : THandle;
  Mode : LongInt;
  bufsize : SizeInt;
  _private : SizeInt;
  bufpos : SizeInt;
  bufend : SizeInt;
  bufptr : ^TextBuf;
  openfunc : pointer;
  inoutfunc : pointer;
  flushfunc : pointer;
  closefunc : pointer;
  UserData : Array[1..32] of Byte;
  name : Array[0..textrecnamelength-1] of Char;
  LineEnd : TLineEndStr;
  buffer : TextBuf;
end
```

`TextRec` describes the internal working of a `Text` file.

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

```
TLineEndStr =
```

`TLineEndStr` is used in the TextRec (491) record to indicate the end-of-line sequence for a text file.

5.9.3 Variables

```
DosError : Integer
```

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

Other values are possible, but are not documented.

Table 5.8: Dos error codes

Value	Meaning
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.

5.10 Procedures and functions

5.10.1 AddDisk

Synopsis: Add disk definition to list if drives (Unix only)

Declaration: `function AddDisk(const path: String) : Byte`

Visibility: default

Description: `AddDisk` adds a filename `S` to the internal list of disks. It is implemented for systems which do not use DOS type drive letters. This list is used to determine which disks to use in the DiskFree (492) and DiskSize (493) calls. The DiskFree (492) and DiskSize (493) functions need a file on the specified drive, since this is required for the `statfs` system call. The names are added sequentially. The dos initialization code presets the first three disks to:

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

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

Errors: None

See also: DiskFree (492), DiskSize (493)

5.10.2 DiskFree

Synopsis: Get free size on Disk.

Declaration: `function DiskFree(drive: Byte) : Int64`

Visibility: default

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.

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

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

There is room for 1-26 drives. You can add a drive with the AddDisk (492) procedure. These settings can be coded in `dos.pp`, in the initialization part.

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

See also: DiskSize (493), AddDisk (492)

Listing: ./dosex/ex6.pp

```
Program Example6;
uses Dos;

{ Program to demonstrate the DiskSize and DiskFree function. }

begin
  WriteLn('This partition size has ',DiskSize(0),' bytes');
  WriteLn('Currently ',DiskFree(0),' bytes are free');
end.
```

5.10.3 DiskSize

Synopsis: Get total size of disk.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

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.

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

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

There is room for 1-26 drives. You can add a drive with the AddDisk (492) procedure. These settings can be coded in `dos.pp`, in the initialization part.

For an example, see DiskFree (492).

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

See also: DiskFree (492), AddDisk (492)

5.10.4 DosExitCode

Synopsis: Exit code of last executed program.

Declaration: `function DosExitCode : Word`

Visibility: default

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

Errors: None.

See also: Exec (496)

Listing: ./dosex/ex5.pp

```
Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

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

5.10.5 DosVersion

Synopsis: Current OS version

Declaration: `function DosVersion : Word`

Visibility: default

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

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

Errors: None.

Listing: ./dosex/ex1.pp

```
Program Example1;
uses Dos;

{ Program to demonstrate the DosVersion function. }

var
  OS       : string[32];
```

```
  Version : word;
begin
{$IFDEF LINUX}
  OS:='Linux';
{$ENDIF}
{$ifdef FreeBSD}
  OS:='FreeBSD';
{$endif}
{$ifdef NetBSD}
  OS:='NetBSD';
{$endif}
{$ifdef Solaris}
  OS:='Solaris';
{$endif}
{$ifdef QNX}
  OS:='QNX';
{$endif}

{$IFDEF DOS}
  OS:='Dos';
{$ENDIF}
  Version:=DosVersion;
  WriteLn('Current ',OS,' version is ',Lo(Version),'.',Hi(Version));
end.
```

5.10.6 DTToUnixDate

Synopsis: Convert a DateTime to unix timestamp

Declaration: `function DTToUnixDate(DT: DateTime) : LongInt`

Visibility: default

Description: `DTToUnixDate` converts the DateTime value in `DT` to a unix timestamp. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: UnixDateToDT (510), PackTime (506), UnpackTime (510), GetTime (504), SetTime (509)

5.10.7 EnvCount

Synopsis: Return the number of environment variables

Declaration: `function EnvCount : LongInt`

Visibility: default

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

Errors: None.

See also: EnvStr (496), GetEnv (501)

5.10.8 EnvStr

Synopsis: Return environment variable by index

Declaration: `function EnvStr(Index: LongInt) : String`

Visibility: default

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

Errors: The length is limited to 255 characters.

See also: EnvCount (495), GetEnv (501)

Listing: ./dosex/ex13.pp

```
Program Example13;
uses Dos;

{ Program to demonstrate the EnvCount and EnvStr function. }

var
  i : Longint;
begin
  WriteLn('Current Environment is:');
  for i:=1 to EnvCount do
   WriteLn(EnvStr(i));
end.
```

5.10.9 Exec

Synopsis: Execute another program, and wait for it to finish.

Declaration: `procedure Exec(const path: PathStr;const comline: ComStr)`

Visibility: default

Description: `Exec` executes the program in `Path`, with the options given by `ComLine`. The program name should *not* appear again in `ComLine`, it is specified in `Path`. Comline contains only the parameters that are passed to the program.

After the program has terminated, the procedure returns. The Exit value of the program can be consulted with the `DosExitCode` function.

For an example, see DosExitCode (494)

Errors: Errors are reported in `DosError`.

See also: DosExitCode (494)

5.10.10 FExpand

Synopsis: Expand a relative path to an absolute path

Declaration: `function FExpand(const path: PathStr) : PathStr`

Visibility: default

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

Remark: On case sensitive file systems (such as unix and linux), the resulting name is left as it is, otherwise it is converted to uppercase.

Errors: FSplit (499)

Listing: ./dosex/ex11.pp

```
Program Example11;
uses Dos;

{ Program to demonstrate the FExpand function. }

begin
  WriteLn('Expanded Name of this program is ',FExpand(ParamStr(0)));
end.
```

5.10.11 FindClose

Synopsis: Dispose resources allocated by a FindFirst (497)/FindNext (498) sequence.

Declaration: `procedure FindClose(var f: SearchRec)`

Visibility: default

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

This call is needed to free any internal resources allocated by the FindFirst (497) or FindNext (498) calls.

The unix 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.

Remark: It is recommended to use the linux call `Glob` when looking for files on linux.

Errors: Errors are reported in DosError.

See also: FindFirst (497), FindNext (498)

5.10.12 FindFirst

Synopsis: Start search for one or more files.

Declaration: `procedure FindFirst(const path: PathStr;attr: Word;var f: SearchRec)`

Visibility: default

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

It returns a `SearchRec` record for further searching in `F`. `Path` can contain the wildcard characters `?` (matches any single character) and `*` (matches 0 ore more arbitrary characters). In this case

FindFirst will return the first file which matches the specified criteria. If DosError is different from zero, no file(s) matching the criteria was(were) found.

Remark: On os/2, you cannot issue two different FindFirst calls. That is, you must close any previous search operation with FindClose (497) before starting a new one. Failure to do so will end in a Run-Time Error 6 (Invalid file handle)

Errors: Errors are reported in DosError.

See also: FindNext (498), FindClose (497)

Listing: ./dosex/ex7.pp

```
Program Example7;
uses Dos;

{ Program to demonstrate the FindFirst and FindNext function. }

var
  Dir : SearchRec;
begin
  FindFirst('*.*',archive,Dir);
  WriteLn('FileName'+Space(32),'FileSize':9);
  while (DosError=0) do
   begin
     WriteIn(Dir.Name+Space(40-Length(Dir.Name)),Dir.Size:9);
     FindNext(Dir);
   end;
  FindClose(Dir);
end.
```

5.10.13 FindNext

Synopsis: Find next matching file after FindFirst (497)

Declaration: `procedure FindNext(var f: SearchRec)`

Visibility: default

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.

For an example, see FindFirst (497).

Errors: DosError is used to report errors.

See also: FindFirst (497), FindClose (497)

5.10.14 FSearch

Synopsis: Search a file in searchpath

Declaration: `function FSearch(path: PathStr;dirlist: String) : PathStr`

Visibility: default

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

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

Errors: None.

See also: FExpand (496)

Listing: ./dosex/ex10.pp

```
program Example10;

uses Dos;

{ Program to demonstrate the FSearch function. }

var s:pathstr;

begin
  s:=FSearch(ParamStr(1),GetEnv('PATH'));
  if s='' then
    WriteLn(ParamStr(1),' not Found in PATH')
  else
    WriteIn(ParamStr(1),' Found in PATH at ',s);
end.
```

5.10.15 FSplit

Synopsis: Split a full-path filename in parts.

Declaration:
```
procedure FSplit(path: PathStr;var dir: DirStr;var name: NameStr;
                 var ext: ExtStr)
```

Visibility: default

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

Errors: None.

See also: FSearch (498)

Listing: ./dosex/ex12.pp

```
program Example12;

uses Dos;

{ Program to demonstrate the FSplit function. }

var dir:dirstr;
    name:namestr;
    ext:extstr;

begin
```

```
  FSplit(ParamStr(1),dir,name,ext);
  WriteLn('Splitted ',ParamStr(1),' in:');
  WriteLn('Path      : ',dir);
  WriteLn('Name      : ',name);
  WriteLn('Extension : ',ext);
end.
```

5.10.16 GetCBreak

Synopsis: Get control-Break flag

Declaration: `procedure GetCBreak(var breakvalue: Boolean)`

Visibility: default

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

Remark: Under non-dos and non-Amiga operating systems, `BreakValue` always returns `True`.

Errors: None

See also: SetCBreak (507)

5.10.17 GetDate

Synopsis: Get the current date

Declaration: `procedure GetDate(var year: Word;var month: Word;var mday: Word;`
`var wday: Word)`

Visibility: default

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 (504), SetDate (507)

Listing: ./dosex/ex2.pp

```
Program Example2;
uses Dos;

{ Program to demonstrate the GetDate function. }

const
  DayStr:array[0..6] of string[3]=('Sun','Mon','Tue','Wed','Thu','Fri','Sat');
  MonthStr:array[1..12] of string[3]=('Jan','Feb','Mar','Apr','May','Jun',
                                      'Jul','Aug','Sep','Oct','Nov','Dec');
var
  Year,Month,Day,WDay : word;
begin
  GetDate(Year,Month,Day,WDay);
  WriteLn('Current date');
  WriteLn(DayStr[WDay],', ',Day,' ',MonthStr[Month],' ',Year,'.');
end.
```

5.10.18 GetEnv

Synopsis: Get environment variable by name.

Declaration: `function GetEnv(envvar: String) : String`

Visibility: default

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

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

Errors: None.

See also: EnvCount (495), EnvStr (496)

Listing: ./dosex/ex14.pp

```
Program Example14;
uses Dos;

{ Program to demonstrate the GetEnv function. }

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

5.10.19 GetFAttr

Synopsis: Get file attributes

Declaration: `procedure GetFAttr(var f;var attr: Word)`

Visibility: default

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

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

Under linux, supported attributes are:

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

Errors: Errors are reported in `DosError`

See also: SetFAttr (507)

Listing: ./dosex/ex8.pp

```
Program Example8;
uses Dos;

{ Program to demonstrate the GetFAttr function. }

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

5.10.20 GetFTime

Synopsis: Get file last modification time.

Declaration: `procedure GetFTime(var f;var time: LongInt)`

Visibility: default

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 (508), PackTime (506), UnPackTime (510)

Listing: ./dosex/ex9.pp

```
Program Example9;
uses Dos;

{ Program to demonstrate the GetFTime function. }

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

var
  f    : File;
  Time : Longint;
  DT   : DateTime;
```

```
begin
  if Paramcount>0 then
    Assign(f,ParamStr(1))
  else
    Assign(f,'ex9.pp');
  Reset(f);
  GetFTime(f,Time);
  Close(f);
  UnPackTime(Time,DT);
  Write ('File ',ParamStr(1),' is last modified on ');
  WriteIn (L0(DT.Month),'-',L0(DT.Day),'-',DT.Year,
           ' at ',L0(DT.Hour),':',L0(DT.Min));
end.
```

5.10.21 GetIntVec

Synopsis: Get interrupt vector

Declaration: `procedure GetIntVec(intno: Byte;var vector: pointer)`

Visibility: default

Description: `GetIntVec` returns the address of interrupt vector `IntNo`.

Remark: This call does nothing, it is present for compatibility only. Modern systems do not allow low level access to the hardware.

Errors: None.

See also: SetIntVec (508)

5.10.22 GetLongName

Synopsis: Get the long filename of a DOS 8.3 filename.

Declaration: `function GetLongName(var p: String) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 and Win32 versions of Free Pascal.

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

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

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

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

See also: GetShortName (504)

5.10.23 GetMsCount

Synopsis: Number of milliseconds since a starting point.

Declaration: `function GetMsCount : Int64`

Visibility: default

Description: `GetMSCount` returns a number of milliseconds elapsed since a certain moment in time. This moment in time is implementation dependent. This function is used for timing purposes: Substracting the results of 2 subsequent calls to this function returns the number of milliseconds elapsed between the two calls.

This call is not very reliable, it is recommended to use some system specific calls for timings.

See also: GetTime (504)

5.10.24 GetShortName

Synopsis: Get the short (8.3) filename of a long filename.

Declaration: `function GetShortName(var p: String) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 and Win32 versions of Free Pascal.

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

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

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

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

See also: GetLongName (503)

5.10.25 GetTime

Synopsis: Return the current time

Declaration:
```
procedure GetTime(var hour: Word;var minute: Word;var second: Word;
                  var sec100: Word)
```

Visibility: default

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

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

Errors: None.

See also: GetDate (500), SetTime (509)

Listing: ./dosex/ex3.pp

```
Program Example3;
uses Dos;

{ Program to demonstrate the GetTime function. }

Function L0(w:word):string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
```

```
    L0:='0'+s
  else
    L0:=s;
end;

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

5.10.26 GetVerify

Synopsis: Get verify flag

Declaration: `procedure GetVerify(var verify: Boolean)`

Visibility: default

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.

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

Errors: None.

See also: SetVerify (509)

5.10.27 Intr

Synopsis: Execute interrupt

Declaration: `procedure Intr(intno: Byte;var regs: Registers)`

Visibility: default

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`.

Remark: Under non-dos operating systems, this call does nothing.

Errors: None.

See also: MSDos (506)

5.10.28 Keep

Synopsis: Terminate and stay resident.

Declaration: `procedure Keep(exitcode: Word)`

Visibility: default

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.

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

Errors: None.

5.10.29 MSDos

Synopsis: Execute MS-DOS system call

Declaration: `procedure MSDos(var regs: Registers)`

Visibility: default

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

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

Errors: None.

See also: Intr (505)

5.10.30 PackTime

Synopsis: Pack DateTime value to a packed-time format.

Declaration: `procedure PackTime(var t: DateTime;var p: LongInt)`

Visibility: default

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 (508), FindFirst (497), FindNext (498), UnPackTime (510)

Listing: ./dosex/ex4.pp

```
Program Example4;
uses Dos;

{ Program to demonstrate the PackTime and UnPackTime functions. }

var
  DT   : DateTime;
  Time : longint;
begin
  with DT do
   begin
     Year:=1998;
     Month:=11;
     Day:=11;
     Hour:=11;
     Min:=11;
     Sec:=11;
   end;
  PackTime(DT,Time);
```

```
  WriteLn('Packed Time : ',Time);
  UnPackTime(Time,DT);
  WriteLn('Unpacked Again:');
  with DT do
   begin
     WriteLn('Year   ',Year);
     WriteLn('Month  ',Month);
     WriteLn('Day    ',Day);
     WriteLn('Hour   ',Hour);
     WriteLn('Min    ',Min);
     WriteLn('Sec    ',Sec);
   end;
end.
```

5.10.31 SetCBreak

Synopsis: Set Control-Break flag status

Declaration: `procedure SetCBreak(breakvalue: Boolean)`

Visibility: default

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

Remark: Under non-dos and non-Amiga operating systems, this call does nothing.

Errors: None.

See also: GetCBreak (500)

5.10.32 SetDate

Synopsis: Set system date

Declaration: `procedure SetDate(year: Word;month: Word;day: Word)`

Visibility: default

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

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

Errors: None.

See also: GetDate (500), SetTime (509)

5.10.33 SetFAttr

Synopsis: Set file attributes

Declaration: `procedure SetFAttr(var f;attr: Word)`

Visibility: default

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

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

Remark: Under unix like systems (such as linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`.

See also: GetFAttr (501)

5.10.34 SetFTime

Synopsis: Set file modification time.

Declaration: `procedure SetFTime(var f;time: LongInt)`

Visibility: default

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.

Remark: Under unix like systems (such as linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`

See also: GetFTime (502), PackTime (506), UnPackTime (510)

5.10.35 SetIntVec

Synopsis: Set interrupt vector

Declaration: `procedure SetIntVec(intno: Byte;vector: pointer)`

Visibility: default

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

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

Errors: None.

See also: GetIntVec (503)

5.10.36 SetTime

Synopsis: Set system time

Declaration: `procedure SetTime(hour: Word;minute: Word;second: Word;sec100: Word)`

Visibility: default

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

Remark: On a linux machine, there must be root privileges, otherwise this routine will do nothing. On other unix systems, this call currently does nothing.

Errors: None.

See also: GetTime (504), SetDate (507)

5.10.37 SetVerify

Synopsis: Set verify flag

Declaration: `procedure SetVerify(verify: Boolean)`

Visibility: default

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.

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

Errors: None.

See also: SetVerify (509)

5.10.38 SwapVectors

Synopsis: Swap interrupt vectors

Declaration: `procedure SwapVectors`

Visibility: default

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.

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

Errors: None.

See also: Exec (496), SetIntVec (508)

5.10.39 UnixDateToDt

Synopsis: Convert a unix timestamp to a DateTime record

Declaration: `procedure UnixDateToDt(SecsPast: LongInt;var Dt: DateTime)`

Visibility: default

Description: `DTToUnixDate` converts the unix timestamp value in `SecsPast` to a DateTime representation in `DT`. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: DTToUnixDate (495), PackTime (506), UnpackTime (510), GetTime (504), SetTime (509)

5.10.40 UnpackTime

Synopsis: Unpack packed file time to a DateTime value

Declaration: `procedure UnpackTime(p: LongInt;var t: DateTime)`

Visibility: default

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.

For an example, see PackTime (506).

Errors: None.

See also: GetFTime (502), FindFirst (497), FindNext (498), PackTime (506)

5.10.41 weekday

Synopsis: Return the day of the week

Declaration: `function weekday(y: LongInt;m: LongInt;d: LongInt) : LongInt`

Visibility: default

Description: `WeekDay` returns the day of the week on which the day `Y/M/D` falls. Sunday is represented by 0, Saturday is 6.

Errors: On error, -1 is returned.

See also: PackTime (506), UnpackTime (510), GetTime (504), SetTime (509)

Chapter 6

Reference for unit 'dxeload'

6.1 Overview

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

It exists only for dos.

6.2 Procedures and functions

6.2.1 dxe_load

Synopsis: Load DXE file in memory

Declaration: `function dxe_load(filename: String) : pointer`

Visibility: default

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

For an example, see the emu387 (515) unit in the RTL.

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

Chapter 7

Reference for unit 'dynlibs'

7.1 Overview

The Dynlibs unit provides support for dynamically loading shared libraries. It is available only on those platforms that support shared libraries. The functionality available here may only be a part of the functionality available on each separate platform, in the interest of portability.

On unix platforms, using this unit will cause the program to be linked to the C library, as most shared libraries are implemented in C and the dynamical linker too.

7.2 Constants, types and variables

7.2.1 Constants

```
NilHandle = TLibHandle ( 0 )
```

Correctly typed Nil handle - returned on error by LoadLibrary (513)

```
SharedSuffix = 'so'
```

7.2.2 Types

```
HModule = TLibHandle
```

Alias for TLibHandle (512) type.

```
TLibHandle = PtrInt
```

`TLibHandle` should be considered an opaque type. It is defined differently on various platforms. The definition shown here depends on the platform for which the documentation was generated.

7.3 Procedures and functions

7.3.1 FreeLibrary

Synopsis: For compatibility with Delphi/Windows: Unload a library

Declaration: `function FreeLibrary(Lib: TLibHandle) : Boolean`

Visibility: default

Description: `FreeLibrary` provides the same functionality as UnloadLibrary (514), and is provided for compatibility with Delphi.

See also: UnloadLibrary (514)

7.3.2 GetProcAddress

Synopsis: For compatibility with Delphi/Windows: Get the address of a procedure

Declaration: `function GetProcAddress(Lib: TLibHandle;ProcName: AnsiString) : Pointer`

Visibility: default

Description: `GetProcAddress` provides the same functionality as GetProcedureAddress (513), and is provided for compatibility with Delphi.

See also: GetProcedureAddress (513)

7.3.3 GetProcedureAddress

Synopsis: Get the address of a procedure or symbol in a dynamic library.

Declaration: `function GetProcedureAddress(Lib: TLibHandle;ProcName: AnsiString) : Pointer`

Visibility: default

Description: `GetProcedureAddress` returns a pointer to the location in memory of the symbol `ProcName` in the dynamically loaded library specified by it's handle `lib`. If the symbol cannot be found or the handle is invalid, `Nil` is returned.

On Windows, only an exported procedure or function can be searched this way. On Unix platforms the location of any exported symbol can be retrieved this way.

Errors: If the symbol cannot be found, `Nil` is returned.

See also: LoadLibrary (513), UnLoadLibrary (514)

7.3.4 LoadLibrary

Synopsis: Load a dynamic library and return a handle to it.

Declaration: `function LoadLibrary(Name: AnsiString) : TLibHandle`

Visibility: default

Description: `LoadLibrary` loads a dynamic library in file `Name` and returns a handle to it. If the library cannot be loaded, NilHandle (512) is returned.

No assumptions should be made about the location of the loaded library if a relative pathname is specified. The behaviour is dependent on the platform. Therefore it is best to specify an absolute pathname if possible.

Errors: On error, NilHandle (512) is returned.

See also: UnloadLibrary (514), GetProcedureAddress (513)

7.3.5 SafeLoadLibrary

Synopsis: Saves the control word and loads a library

Declaration: `function SafeLoadLibrary(Name: AnsiString) : TLibHandle`

Visibility: default

Description: `SafeLoadLibary` saves the FPU control word, and calls LoadLibrary (513) with library name `Name`. After that function has returned, the FPU control word is saved again. (only on Intel i386 CPUS).

See also: LoadLibrary (513)

7.3.6 UnloadLibrary

Synopsis: Unload a previously loaded library

Declaration: `function UnloadLibrary(Lib: TLibHandle) : Boolean`

Visibility: default

Description: `UnloadLibrary` unloads a previously loaded library (specified by the handle `lib`). The call returns `True` if succesful, `False` otherwisa.

Errors: On error, `False` is returned.

See also: LoadLibrary (513), GetProcedureAddress (513)

Chapter 8

Reference for unit 'emu387'

8.1 Overview

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

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

8.2 Procedures and functions

8.2.1 npxsetup

Synopsis: Set up coprocessor emulation.

Declaration: `procedure npxsetup(prog_name: String)`

Visibility: default

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

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

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

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

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

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

Chapter 9

Reference for unit 'getopts'

9.1 Overview

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

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

9.2 Constants, types and variables

9.2.1 Constants

```
EndOfOptions = #255
```

Returned by getopt (518),getlongopts (518) to indicate that there are no more options.

```
No_Argument = 0
```

Specifies that a long option does not take an argument.

```
Optional_Argument = 2
```

Specifies that a long option optionally takes an argument.

```
OptSpecifier : Set of Char = ['-']
```

Character indicating an option on the command-line.

```
Required_Argument = 1
```

Specifies that a long option needs an argument.

Table 9.1: Enumeration values for type Orderings

Value	Explanation
permute	Change command-line options.
require_order	Don't touch the ordering of the command-line options
return_in_order	Return options in the correct order.

9.2.2 Types

```
Orderings = (require_order,permute,return_in_order)
```

Command-line ordering options.

```
POption = ^TOption
```

Pointer to TOption (517) record.

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

The `TOption` type is used to communicate the long options to GetLongOpts (518). 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`.

9.2.3 Variables

```
OptArg : String
```

Set to the argument of an option, if the option needs one.

```
OptErr : Boolean
```

Indicates whether `getopt()` prints error messages.

```
OptInd : LongInt
```

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`.

```
OptOpt : Char
```

In case of an error, contains the character causing the error.

9.3 Procedures and functions

9.3.1 GetLongOpts

Synopsis: Return next long option.

Declaration:
```
function GetLongOpts(ShortOpts: String;LongOpts: POption;
                     var Longind: LongInt) : Char
```

Visibility: default

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 (518) 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 (518).

See also: Getopt (518)

9.3.2 GetOpt

Synopsis: Get next short option.

Declaration:
```
function GetOpt(ShortOpts: String) : Char
```

Visibility: default

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 (518)

Listing: ./optex/optex.pp

```
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 10

Reference for unit 'go32'

10.1 Real mode callbacks

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

- interrupts disabled
- `%CS:%EIP` = 48 bit pointer specified in the original call to get_rm_callback (539)
- `%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 (539), free_rm_callback (535), lock_code (547), lock_data (548)

10.2 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 (557) and it's linear address via transfer_buffer (557). Another way is to allocate a completely new dos memory area via the global_dos_alloc (544) function for your use and supply its real mode address.

See also: tb_size (557), transfer_buffer (557), global_dos_alloc (544), global_dos_free (546), realintr (550)

10.3 Software interrupts

Ordinarily, a handler installed with set_pm_interrupt (554) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by set_rm_interrupt (555).

See also: set_rm_interrupt (555), get_rm_interrupt (542), set_pm_interrupt (554), get_pm_interrupt (539), lock_data (548), lock_code (547), enable (534), disable (532), outportb (549)

10.4 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.

10.5 Disabling interrupts

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

10.6 Creating your 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.

10.7 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.

10.8 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.

10.9 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. Functions that handle hardware interrupts are called handlers.

10.10 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 (535), get_ds (536), get_ss (544)

10.11 I/O port access

The I/O port access is done via the various inportb (546), outportb (549) 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 (549), inportb (546)

10.12 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 (526), dosmemput (526), dosmemmove (526), dosmemfillchar (525), dosmemfillword (525), seg_move (553), seg_fillchar (551), seg_fillword (552)

10.13 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 (535), get_ds (536), get_ss (544), allocate_ldt_descriptors (528), free_ldt_descriptor (534), segment_to_descriptor (551), get_next_selector_increment_value (538), get_segment_base_address (543), set_segment_base_address (555), set_segment_limit (556), create_code_segment_alias_descriptor (532)

10.14 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.

10.15 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.

10.16 Overview

This document describes 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.

Only the GO32V2 DPMI mode is discussed by me here due to the fact that new applications shouldn't be created with the older GO32V1 model. The go32v2 version is much more advanced and better. Additionally a lot of functions only work in DPMI mode anyway. I hope the following explanations and introductions aren't too confusing at all. If you notice an error or bug send it to the FPC mailing list or directly to me. So let's get started and happy and error free coding I wish you.... Thomas Schatzl, 25. August 1998

10.17 Constants, types and variables

10.17.1 Constants

```
auxcarryflag = $010
```

Check for auxiliary carry flag in trealregs (528)

```
carryflag = $001
```

Check for carry flag in trealregs (528)

```
directionflag = $400
```

Check for direction flag in trealregs (528)

```
dosmemfillchar : procedure(seg: Word;ofs: Word;count: LongInt;c: Char) = @dpmi_dosmem
```

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.

```
dosmemfillword : procedure(seg: Word;ofs: Word;count: LongInt;w: Word) = @dpmi_dosmem
```

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.

```
dosmemget : procedure(seg: Word;ofs: Word;var data;count: LongInt) = @dpmi_dosmemget
```

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.
For an example, see global_dos_alloc (544).

```
dosmemmove : procedure(sseg: Word;sofs: Word;dseg: Word;dofs: Word;count: LongInt) =
```

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.

```
dosmemput : procedure(seg: Word;ofs: Word;var data;count: LongInt) = @dpmi_dosmemput
```

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.
For an example, see global_dos_alloc (544).

```
interruptflag = $200
```

Check for interrupt flag in trealregs (528)

```
overflowflag = $800
```

Check for overflow flag in trealregs (528)

```
parityflag = $004
```

Check for parity flag in trealregs (528)

```
rm_dpmi = 4
```

get_run_mode (543) return value: DPMI (e.g. dos box or 386Max)

```
rm_raw = 1
```

get_run_mode (543) return value: raw (without HIMEM)

```
rm_unknown = 0
```

get_run_mode (543) return value: Unknown runmode

```
rm_vcpi = 3
```

get_run_mode (543) return value: VCPI (with HIMEM and EMM386)

```
rm_xms = 2
```

get_run_mode (543) return value: XMS (with HIMEM, without EMM386)

```
signflag = $080
```

Check for sign flag in trealregs (528)

```
trapflag = $100
```

Check for trap flag in trealregs (528)

```
zeroflag = $040
```

Check for zero flag in trealregs (528)

10.17.2 Types

```
registers = trealregs
```

Alias for trealregs (528)

```
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;
  reserved0 : LongInt;
  reserved1 : LongInt;
  reserved2 : LongInt;
end
```

`tmeminfo` Holds information about the memory allocation, etc.

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.

```
trealregs = record
end
```

The `trealregs` type contains the data structure to pass register values to a interrupt handler or real mode callback.

```
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

10.17.3 Variables

```
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.

```
int31error : Word
```

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

10.18 Procedures and functions

10.18.1 allocate_ldt_descriptors

Synopsis: Allocate a number of descriptors

Declaration: `function allocate_ldt_descriptors(count: Word) : Word`

Visibility: default

Description: Allocates a number of new descriptors.

Parameters:

count:\ specifies the number of requested unique descriptors.

Return value: The base selector.

Remark: 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 (538) function.

Errors: Check the int31error (528) variable.

See also: free_ldt_descriptor (534), get_next_selector_increment_value (538), segment_to_descriptor (551), create_code_segment_alias_descriptor (532), set_segment_limit (556), set_segment_base_address (555)

Listing: ./go32ex/seldes.pp

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

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

        linB8000 = $B800 * 16;

type
        string80 = string[80];

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

        text_sel : Word;

procedure status(s : string80);
begin
     gotoxy(1, 1); 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.
```

10.18.2 allocate_memory_block

Synopsis: Allocate a block of linear memory

Declaration: `function allocate_memory_block(size: LongInt) : LongInt`

Visibility: default

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.

Remark: *warning* According to my DPMI docs this function is not implemented correctly. Normally you should also get a blockhandle to this block after successful operation. This handle can then be used to free the memory block afterwards or use this handle for other purposes. Since the function isn't implemented correctly, and doesn't return a blockhandle, the block can't be deallocated and is hence 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 (528) variable.

See also: free_memory_block (534)

10.18.3 copyfromdos

Synopsis: Copy data from DOS to to heap

Declaration: `procedure copyfromdos(var addr;len: LongInt)`

Visibility: default

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

Parameters:

addrdata to copy to.

lennumber of bytes to copy to heap.

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

Errors: Check the int31error (528) variable.

See also: tb_size (557), transfer_buffer (557), copytodos (531)

10.18.4 copytodos

Synopsis: Copy data from heap to DOS memory

Declaration: `procedure copytodos(var addr;len: LongInt)`

Visibility: default

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

Parameters:

addrdata to copy from.

lennumber 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 (528) variable.

See also: tb_size (557), transfer_buffer (557), copyfromdos (531)

10.18.5 create_code_segment_alias_descriptor

Synopsis: Create new descriptor from existing descriptor

Declaration: `function create_code_segment_alias_descriptor(seg: Word) : Word`

Visibility: default

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

Parameters:

segDescriptor.

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 (528) variable.

See also: allocate_ldt_descriptors (528), set_segment_limit (556), set_segment_base_address (555)

10.18.6 disable

Synopsis: Disable hardware interrupts

Declaration: `procedure disable`

Visibility: default

Description: Disables all hardware interrupts by execution a CLI instruction.

Errors: None.

See also: enable (534)

10.18.7 dpmi_dosmemfillchar

Synopsis: Fill DOS memory with a character

Declaration:
```
procedure dpmi_dosmemfillchar(seg: Word;ofs: Word;count: LongInt;
                              c: Char)
```

Visibility: default

Description: `dpmi_dosmemfillchar` fills the DOS memory reagion indicated by `seg`,`ofs` with `count` characters `c`.

See also: dpmi_dosmemput (533), dpmi_dosmemget (533), dpmi_dosmemmove (533), dpmi_dosmemfillword (533)

10.18.8 dpmi_dosmemfillword

Synopsis: Fill DOS memory with a word value

Declaration:
```
procedure dpmi_dosmemfillword(seg: Word;ofs: Word;count: LongInt;
                              w: Word)
```

Visibility: default

Description: `dpmi_dosmemfillword` fills the DOS memory reagion indicated by `seg`,`ofs` with `count` words `W`.

See also: dpmi_dosmemput (533), dpmi_dosmemget (533), dpmi_dosmemfillchar (532), dpmi_dosmemmove (533)

10.18.9 dpmi_dosmemget

Synopsis: Move data from DOS memory to DPMI memory

Declaration:
```
procedure dpmi_dosmemget(seg: Word;ofs: Word;var data;count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemput` moves `count` bytes of data from the DOS memory location indicated by `seg` and `ofs` to DPMI memory indicated by `data`.

See also: dpmi_dosmemput (533), dpmi_dosmemmove (533), dpmi_dosmemfillchar (532), dpmi_dosmemfillword (533)

10.18.10 dpmi_dosmemmove

Synopsis: Move DOS memory

Declaration:
```
procedure dpmi_dosmemmove(sseg: Word;sofs: Word;dseg: Word;dofs: Word;
                          count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemmove` moves `count` bytes from DOS memory `sseg`,`sofs` to `dseg`,`dofs`.

See also: dpmi_dosmemput (533), dpmi_dosmemget (533), dpmi_dosmemfillchar (532), dpmi_dosmemfillword (533)

10.18.11 dpmi_dosmemput

Synopsis: Move data from DPMI memory to DOS memory.

Declaration:
```
procedure dpmi_dosmemput(seg: Word;ofs: Word;var data;count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemput` moves `count` bytes of data from `data` to the DOS memory location indicated by `seg` and `ofs`.

See also: dpmi_dosmemget (533), dpmi_dosmemmove (533), dpmi_dosmemfillchar (532), dpmi_dosmemfillword (533)

10.18.12 enable

Synopsis: Enable hardware interrupts

Declaration: `procedure enable`

Visibility: default

Description: Enables all hardware interrupts by executing a STI instruction.

Errors: None.

See also: disable (532)

10.18.13 free_ldt_descriptor

Synopsis: Free a descriptor

Declaration: `function free_ldt_descriptor(d: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated descriptor.

Parameters:

desThe 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 (528) must be freed individually with this function, even if it was previously allocated as a part of a contiguous array of descriptors.

For an example, see allocate_ldt_descriptors (528).

Errors: Check the int31error (528) variable.

See also: allocate_ldt_descriptors (528), get_next_selector_increment_value (538)

10.18.14 free_memory_block

Synopsis: Free allocated memory block

Declaration: `function free_memory_block(blockhandle: LongInt) : Boolean`

Visibility: default

Description: Frees a previously allocated memory block.

Parameters:

blockhandlethe 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 (531) . This function doesn't free any descriptors mapped to this block, it's the application's responsibility.

Errors: Check int31error (528) variable.

See also: allocate_memory_block (531)

10.18.15 free_rm_callback

Synopsis: Release real mode callback.

Declaration: `function free_rm_callback(var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Releases a real mode callback address that was previously allocated with the get_rm_callback (539) function.

Parameters:

intaddrreal mode address buffer returned by get_rm_callback (539) .

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

For an example, see get_rm_callback (539).

Errors: Check the int31error (528) variable.

See also: set_rm_interrupt (555), get_rm_callback (539)

10.18.16 get_cs

Synopsis: Get CS selector

Declaration: `function get_cs : Word`

Visibility: default

Description: Returns the cs selector.

Return value: The content of the cs segment register.

For an example, see set_pm_interrupt (554).

Errors: None.

See also: get_ds (536), get_ss (544)

10.18.17 get_descriptor_access_right

Synopsis: Get descriptor's access rights

Declaration: `function get_descriptor_access_right(d: Word) : LongInt`

Visibility: default

Description: Gets the access rights of a descriptor.

Parameters:

dselector to descriptor.

Return value: Access rights bit field.

Errors: Check the int31error (528) variable.

See also: set_descriptor_access_right (553)

10.18.18 get_ds

Synopsis: Get DS Selector

Declaration: `function get_ds : Word`

Visibility: default

Description: Returns the ds selector.

Return values: The content of the ds segment register.

Errors: None.

See also: get_cs (535), get_ss (544)

10.18.19 get_exception_handler

Synopsis: Return current exception handler

Declaration: `function get_exception_handler(e: Byte;var intaddr: tseginfo) : Boolean`

Visibility: default

Description: `get_exception_handler` returns the exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: set_exception_handler (553), get_pm_exception_handler (539)

10.18.20 get_linear_addr

Synopsis: Convert physical to linear address

Declaration: `function get_linear_addr(phys_addr: LongInt;size: LongInt) : LongInt`

Visibility: default

Description: Converts a physical address into a linear address.

Parameters:

phys_addrphysical address of device.

sizeSize 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 (528) variable.

See also: allocate_ldt_descriptors (528), set_segment_limit (556), set_segment_base_address (555)

10.18.21 get_meminfo

Synopsis: Return information on the available memory

Declaration: `function get_meminfo(var meminfo: tmeminfo) : Boolean`

Visibility: default

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

Parameters:

meminfobuffer to fill memory information into.

Return values: Due to an implementation bug this function always returns `False`, but it always succeeds.

Remark: 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 (538) function.

Errors: Check the int31error (528) variable.

See also: get_page_size (538)

Listing: ./go32ex/meminfo.pp

```
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 begin
                with meminfo do begin
                        Writeln('Largest available free block : ',
                                available_memory div 1024, ' kbytes');
                        if (available_pages <> -1) then
                                Writeln('Maximum available unlocked pages : ',
                                        available_pages);
                        if (available_lockable_pages <> -1) then
                                Writeln('Maximum lockable available pages : ',
                                        available_lockable_pages);
                        if (linear_space <> -1) then
                                Writeln('Linear address space size : ',
                                        linear_space*get_page_size div 1024, ' kbytes');
                        if (unlocked_pages <> -1) then
                                Writeln('Total number of unlocked pages : ',
                                        unlocked_pages);
                        if (available_physical_pages <> -1) then
                                Writeln('Total number of free pages : ',
                                        available_physical_pages);
                        if (total_physical_pages <> -1) then
                                Writeln('Total number of physical pages : ',
```

```
                                        total_physical_pages);
                        if (free_linear_space <> -1) then
                                WriteLn('Free linear address space : ',
                                        free_linear_space*get_page_size div 1024,
                                        ' kbytes');
                        if (max_pages_in_paging_file <> -1) then
                                WriteLn('Maximum size of paging file : ',
                                        max_pages_in_paging_file*get_page_size div 1024,
                                        ' kbytes');
                end;
        end;
end.
```

10.18.22 get_next_selector_increment_value

Synopsis: Return selector increment value

Declaration: `function get_next_selector_increment_value : Word`

Visibility: default

Description: Returns the selector increment value when allocating multiple subsequent descriptors via allocate_ldt_descriptors (528).

Return value: Selector increment value.

Remark: Notes: Because allocate_ldt_descriptors (528) 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 (528) variable.

See also: allocate_ldt_descriptors (528), free_ldt_descriptor (534)

10.18.23 get_page_size

Synopsis: Return the page size

Declaration: `function get_page_size : LongInt`

Visibility: default

Description: Returns the size of a single memory page.

Return value: Size of a single page in bytes.

Remark: The returned size is typically 4096 bytes.

For an example, see get_meminfo (537).

Errors: Check the int31error (528) variable.

See also: get_meminfo (537)

10.18.24 get_pm_exception_handler

Synopsis: Get protected mode exception handler

Declaration:
```
function get_pm_exception_handler(e: Byte;var intaddr: tseginfo)
                                        : Boolean
```

Visibility: default

Description: `get_pm_exception_handler` returns the protected mode exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: get_exception_handler (536), set_pm_exception_handler (554)

10.18.25 get_pm_interrupt

Synopsis: Return protected mode interrupt handler

Declaration:
```
function get_pm_interrupt(vector: Byte;var intaddr: tseginfo) : Boolean
```

Visibility: default

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

Parameters:

vectorinterrupt handler number you want the address to.

intaddrbuffer to store address.

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

Remark: The returned address is a protected mode selector:offset address.

For an example, see set_pm_interrupt (554).

Errors: Check the int31error (528) variable.

See also: set_pm_interrupt (554), set_rm_interrupt (555), get_rm_interrupt (542)

10.18.26 get_rm_callback

Synopsis: Return real mode callback

Declaration:
```
function get_rm_callback(pm_func: pointer;const reg: trealregs;
                         var rmcb: tseginfo) : Boolean
```

Visibility: default

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_funcpointer to the protected mode callback function.

regsupplied registers structure.

rmcbbuffer to real mode address of callback function.

Return values: `True` if successful, otherwise `False`.

Remark: 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 (528) variable.

See also: free_rm_callback (535)

Listing: ./go32ex/callback.pp

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

uses
        crt,
        go32;

const
        mouseint = $33;

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

var
        mouse_numbuttons : longint;

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

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

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

    cmpl $1, USERPROC_INSTALLED
    jne .LNoCallback
    pushal
    movw DOSmemSELECTOR, %ax
    movw %ax, %fs
    call *USERPROC_PROC
    popal
.LNoCallback:

    popl %eax
    popw %ds

    pushl %eax
```

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

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

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

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

        lock_data(mouse_regs, sizeof(mouse_regs));
        lock_data(mouse_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
                gotoxy(1, wherey);
                write('MouseX : ', mouse_x:2, ' MouseY : ', mouse_y:2,
                        ' Buttons : ', mouse_b:2);
        end;
        remove_mouse;
end.
```

10.18.27 get_rm_interrupt

Synopsis: Get real mode interrupt vector

Declaration: `function get_rm_interrupt(vector: Byte;var intaddr: tseginfo) : Boolean`

Visibility: default

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

Parameters:

vectorinterrupt vector number.

intaddrbuffer to store real mode `segment:offset` address.

Return values: `True` if successful, `False` otherwise.

Remark: The returned address is a real mode segment address, which isn't valid in protected mode.

Errors: Check the int31error (528) variable.

See also: set_rm_interrupt (555), set_pm_interrupt (554), get_pm_interrupt (539)

10.18.28 get_run_mode

Synopsis: Return current run mode

Declaration: `function get_run_mode : Word`

Visibility: default

Description: Returns the current mode your application runs with.

Return values: One of the constants used by this function.

Errors: None.

See also: get_run_mode (543)

Listing: ./go32ex/getrunmd.pp

```
uses
        go32;

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

10.18.29 get_segment_base_address

Synopsis: Return base address from descriptor table

Declaration: `function get_segment_base_address(d: Word) : LongInt`

Visibility: default

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

Parameters:

dselector of the descriptor you want the base address of.

Return values: Linear base address of specified descriptor.

For an example, see allocate_ldt_descriptors (528).

Errors: Check the int31error (528) variable.

See also: allocate_ldt_descriptors (528), set_segment_base_address (555), allocate_ldt_descriptors (528), set_segment_limit (556), get_segment_limit (544)

10.18.30 get_segment_limit

Synopsis: Return segment limite from descriptor

Declaration: `function get_segment_limit(d: Word) : LongInt`

Visibility: default

Description: Returns a descriptors segment limit.

Parameters:

dselector.

Return value: Limit of the descriptor in bytes.

Errors: Returns zero if descriptor is invalid.

See also: allocate_ldt_descriptors (528), set_segment_limit (556), set_segment_base_address (555), get_segment_base_address (543)

10.18.31 get_ss

Synopsis: Return SS selector

Declaration: `function get_ss : Word`

Visibility: default

Description: Returns the ss selector.

Return values: The content of the ss segment register.

Errors: None.

See also: get_ds (536), get_cs (535)

10.18.32 global_dos_alloc

Synopsis: Allocate DOS real mode memory

Declaration: `function global_dos_alloc(bytes: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of dos real mode memory.

Parameters:

bytessize 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 (546) call.

Errors: Check the int31error (528) variable.

See also: global_dos_free (546)

Listing: ./go32ex/buffer.pp

```
uses
        go32;

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

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

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

var
        selector,
        segment : Word;

        r : trealregs;
        infobuf : VBEInfoBuf;

begin
        fillchar(r, sizeof(r), 0);
        fillchar(infobuf, sizeof(VBEInfoBuf), 0);
        dosalloc(selector, segment, sizeof(VBEInfoBuf));
        if (int31error<>0) then begin
                Writeln('Error while allocating real mode memory, halting');
                halt;
        end;
        infobuf.Signature := 'VBE2';
        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), ' detected');
        end;
end.
```

10.18.33 global_dos_free

Synopsis: Free DOS memory block

Declaration: `function global_dos_free(selector: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated dos memory block.

Parameters:

selectorselector to the dos memory block.

Return value: `True` if successful, `False` otherwise.

Remark: 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 (544).

For an example, see global_dos_alloc (544).

Errors: Check the int31error (528) variable.

See also: global_dos_alloc (544)

10.18.34 inportb

Synopsis: Read byte from I/O port

Declaration: `function inportb(port: Word) : Byte`

Visibility: default

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

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportb (549), inportw (547), inportl (547)

10.18.35 inportl

Synopsis: Read longint from I/O port

Declaration: `function inportl(port: Word) : LongInt`

Visibility: default

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

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportb (549), inportb (546), inportw (547)

10.18.36 inportw

Synopsis: Read word from I/O port

Declaration: `function inportw(port: Word) : Word`

Visibility: default

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

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportw (550), inportb (546), inportl (547)

10.18.37 lock_code

Synopsis: Lock code memory range

Declaration: `function lock_code(functionaddr: pointer;size: LongInt) : Boolean`

Visibility: default

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

Parameters:

functionaddraddress of the function to be locked.

sizesize in bytes to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (539).

Errors: Check the int31error (528) variable.

See also: lock_linear_region (548), lock_data (548), unlock_linear_region (558), unlock_data (558), unlock_code (557)

10.18.38 lock_data

Synopsis: Lock data memory range

Declaration:
```
function lock_data(var data;size: LongInt) : Boolean
```

Visibility: default

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

Parameters:

dataaddress of data to be locked.

sizelength of data to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (539).

Errors: Check the int31error (528) variable.

See also: lock_linear_region (548), lock_code (547), unlock_linear_region (558), unlock_data (558), unlock_code (557)

10.18.39 lock_linear_region

Synopsis: Lock linear memory region

Declaration:
```
function lock_linear_region(linearaddr: LongInt;size: LongInt) : Boolean
```

Visibility: default

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

Parameters:

linearaddrthe linear address of the memory are to be locked.

sizesize in bytes to be locked.

Return value: `True` if successful, False otherwise.

Errors: Check the int31error (528) variable.

See also: lock_data (548), lock_code (547), unlock_linear_region (558), unlock_data (558), unlock_code (557)

10.18.40 map_device_in_memory_block

Synopsis: Map a device into program's memory space

Declaration:
```
function map_device_in_memory_block(handle: LongInt;offset: LongInt;
                                    pagecount: LongInt;device: LongInt)
                                     : Boolean
```

Visibility: default

Description: `map_device_in_memory_block` allows to map a device in memory. This function is a direct call of the extender. For more information about it's arguments, see the extender documentation.

10.18.41 outportb

Synopsis: Write byte to I/O port

Declaration: `procedure outportb(port: Word;data: Byte)`

Visibility: default

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

Parameters:

portthe I/O port number to send data to.

datavalue sent to I/O port.

Return values: None.

Errors: None.

See also: inportb (546), outportl (549), outportw (550)

Listing: ./go32ex/outport.pp

```
uses
        crt,
        go32;

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

10.18.42 outportl

Synopsis: Write longint to I/O port

Declaration: `procedure outportl(port: Word;data: LongInt)`

Visibility: default

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

Parameters:

portthe I/O port number to send data to.

datavalue sent to I/O port.

Return values: None.

For an example, see outportb (549).

Errors: None.

See also: inportl (547), outportw (550), outportb (549)

10.18.43 outportw

Synopsis: Write word to I/O port

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

Visibility: default

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

Parameters:

portthe I/O port number to send data to.
datavalue sent to I/O port.

Return values: None.

For an example, see outportb (549).

Errors: None.

See also: inportw (547), outportl (549), outportb (549)

10.18.44 realintr

Synopsis: Simulate interrupt

Declaration: `function realintr(intnr: Word;var regs: trealregs) : Boolean`

Visibility: default

Description: Simulates an interrupt in real mode.

Parameters:

intnrinterrupt number to issue in real mode.
regsregisters 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.

Remark: 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 (528) variable.

Listing: ./go32ex/flags.pp

```
uses
        go32;

var
        r : trealregs;

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

10.18.45 request_linear_region

Synopsis: Request linear address region.

Declaration:
```
function request_linear_region(linearaddr: LongInt;size: LongInt;
                               var blockhandle: LongInt) : Boolean
```

Visibility: default

Description: `request_linear_region` requests a linear range of addresses of size `Size`, starting at `linearaddr`. If successful, `True` is returned, and a handle to the address region is returned in `blockhandle`.

Errors: On error, `False` is returned.

10.18.46 segment_to_descriptor

Synopsis: Map segment address to descriptor

Declaration:
```
function segment_to_descriptor(seg: Word) : Word
```

Visibility: default

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:

segthe real mode segment you want the descriptor to.

Return values: Descriptor to real mode segment address.

Remark: 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 (528) and change the base address as necessary.

For an example, see seg_fillchar (551).

Errors: Check the int31error (528) variable.

See also: allocate_ldt_descriptors (528), free_ldt_descriptor (534), set_segment_base_address (555)

10.18.47 seg_fillchar

Synopsis: Fill segment with byte value

Declaration:
```
procedure seg_fillchar(seg: Word;ofs: LongInt;count: LongInt;c: Char)
```

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

segselector to memory area.

ofsoffset to memory.

countnumber of bytes to set.

cbyte data which is set.

Return values: None.

Notes: No range check is done in any way.

Errors: None.

See also: seg_move (553), seg_fillword (552), dosmemfillchar (525), dosmemfillword (525), dosmemget (526), dosmemput (526), dosmemmove (526)

Listing: ./go32ex/vgasel.pp

```
uses
        go32;

var
        vgasel : Word;
        r : trealregs;

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

10.18.48 seg_fillword

Synopsis: Fill segment with word value

Declaration: `procedure seg_fillword(seg: Word;ofs: LongInt;count: LongInt;w: Word)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

segselector to memory area.

ofsoffset to memory.

countnumber of words to set.

wword data which is set.

Return values: None.

Notes: No range check is done in any way.

For an example, see allocate_ldt_descriptors (528).

Errors: None.

See also: seg_move (553), seg_fillchar (551), dosmemfillchar (525), dosmemfillword (525), dosmemget (526), dosmemput (526), dosmemmove (526)

10.18.49 seg_move

Synopsis: Move data between 2 locations

Declaration:
```
procedure seg_move(sseg: Word;source: LongInt;dseg: Word;dest: LongInt;
                   count: LongInt)
```

Visibility: default

Description: Copies data between two memory locations.

Parameters:

ssegsource selector.

sourcesource offset.

dsegdestination selector.

destdestination offset.

countsize in bytes to copy.

Return values: None.

Remark: Overlapping is only checked if the source selector is equal to the destination selector. No range check is done.

For an example, see allocate_ldt_descriptors (528).

Errors: None.

See also: seg_fillchar (551), seg_fillword (552), dosmemfillchar (525), dosmemfillword (525), dosmemget (526), dosmemput (526), dosmemmove (526)

10.18.50 set_descriptor_access_right

Synopsis: Set access rights to memory descriptor

Declaration:
```
function set_descriptor_access_right(d: Word;w: Word) : LongInt
```

Visibility: default

Description: `set_descriptor_access_right` sets the access rights for descriptor `d` to `w`

10.18.51 set_exception_handler

Synopsis: Set exception handler

Declaration:
```
function set_exception_handler(e: Byte;const intaddr: tseginfo)
                                        : Boolean
```

Visibility: default

Description: `set_exception_handler` sets the exception handler for exception `E` to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: get_exception_handler (536), set_pm_exception_handler (554)

10.18.52 set_pm_exception_handler

Synopsis: Set protected mode exception handler

Declaration:
```
function set_pm_exception_handler(e: Byte;const intaddr: tseginfo)
                                        : Boolean
```

Visibility: default

Description: `set_pm_exception_handler` sets the protected mode exception handler for exception `E` to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: set_exception_handler (553), get_pm_exception_handler (539)

10.18.53 set_pm_interrupt

Synopsis: Set protected mode interrupt handler

Declaration:
```
function set_pm_interrupt(vector: Byte;const intaddr: tseginfo)
                                : Boolean
```

Visibility: default

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

Parameters:

vectornumber of protected mode interrupt to set.

intaddrselector:offset address to the interrupt vector.

Return values: `True` if successful, `False` otherwise.

Remark: The address supplied must be a valid `selector:offset` protected mode address.

Errors: Check the int31error (528) variable.

See also: get_pm_interrupt (539), set_rm_interrupt (555), get_rm_interrupt (542)

Listing: ./go32ex/intpm.pp

```
uses
        crt,
        go32;

const
        int1c = $1c;

var
        oldint1c : tseginfo;
        newint1c : tseginfo;

        int1c_counter : Longint;

        int1c_ds : Word; external name '___v2prt0_ds_alias';

procedure int1c_handler; assembler;
asm
    cli
    pushw %ds
    pushw %ax
```

```
    movw %cs:int1c_ds, %ax
    movw %ax, %ds
    incl int1c_counter
    popw %ax
    popw %ds
    sti
    iret
end;

var i : Longint;

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

10.18.54 set_rm_interrupt

Synopsis: Set real mode interrupt handler

Declaration:
```
function set_rm_interrupt(vector: Byte;const intaddr: tseginfo)
                                 : Boolean
```

Visibility: default

Description: Sets a real mode interrupt handler.

Parameters:

vectorthe interrupt vector number to set.

intaddraddress of new interrupt vector.

Return values: `True` if successful, otherwise `False`.

Remark: 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 (539).

Errors: Check the int31error (528) variable.

See also: get_rm_interrupt (542), set_pm_interrupt (554), get_pm_interrupt (539), get_rm_callback (539)

10.18.55 set_segment_base_address

Synopsis: Set descriptor's base address

Declaration:
```
function set_segment_base_address(d: Word;s: LongInt) : Boolean
```

Visibility: default

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

Parameters:

dselector.

snew base address of the descriptor.

Errors: Check the int31error (528) variable.

See also: allocate_ldt_descriptors (528), get_segment_base_address (543), allocate_ldt_descriptors (528), set_segment_limit (556), get_segment_base_address (543), get_segment_limit (544)

10.18.56 set_segment_limit

Synopsis: Set descriptor limit

Declaration: `function set_segment_limit(d: Word;s: LongInt) : Boolean`

Visibility: default

Description: Sets the limit of a descriptor.

Parameters:

dselector.

snew limit of the descriptor.

Return values: Returns `True` if successful, else `False`.

Remark: 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.

For an example, see allocate_ldt_descriptors (528).

Errors: Check the int31error (528) variable.

See also: allocate_ldt_descriptors (528), set_segment_base_address (555), get_segment_limit (544), set_segment_limit (556)

10.18.57 tb_offset

Synopsis: Return DOS transfer buffer offset

Declaration: `function tb_offset : LongInt`

Visibility: default

Description: `tb_offset` returns the DOS transfer buffer segment.

See also: transfer_buffer (557), tb_segment (556), tb_size (557)

10.18.58 tb_segment

Synopsis: Return DOS transfer buffer segment

Declaration: `function tb_segment : LongInt`

Visibility: default

Description: `tb_segment` returns the DOS transfer buffer segment.

See also: transfer_buffer (557), tb_offset (556), tb_size (557)

10.18.59 tb_size

Synopsis: Return DOS transfer memory buffer size

Declaration: `function tb_size : LongInt`

Visibility: default

Description: Returns the size of the pre-allocated dos memory buffer.

Return values: The size of the pre-allocated dos memory buffer. This block always seems to be 16k in size, but don't rely on this.

Errors: None.

See also: transfer_buffer (557), copyfromdos (531), copytodos (531)

10.18.60 transfer_buffer

Synopsis: Return offset of DOS transfer buffer

Declaration: `function transfer_buffer : LongInt`

Visibility: default

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

Errors: None.

See also: tb_size (557)

10.18.61 unlock_code

Synopsis: Unlock code segment

Declaration: `function unlock_code(functionaddr: pointer;size: LongInt) : Boolean`

Visibility: default

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

Parameters:

functionaddraddress of function to be unlocked.

sizesize bytes to be unlocked.

Return value: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (539).

Errors: Check the int31error (528) variable.

See also: unlock_linear_region (558), unlock_data (558), lock_linear_region (548), lock_data (548), lock_code (547)

10.18.62 unlock_data

Synopsis: Unlock data segment

Declaration:
```
function unlock_data(var data;size: LongInt) : Boolean
```

Visibility: default

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

Parameters:

dataaddress of memory to be unlocked.

sizesize bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (539).

Errors: Check the int31error (528) variable.

See also: unlock_linear_region (558), unlock_code (557), lock_linear_region (548), lock_data (548), lock_code (547)

10.18.63 unlock_linear_region

Synopsis: Unlock linear memory region

Declaration:
```
function unlock_linear_region(linearaddr: LongInt;size: LongInt)
                                    : Boolean
```

Visibility: default

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

Parameters:

linearaddrlinear address of the memory to be unlocked.

sizesize bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

Errors: Check the int31error (528) variable.

See also: unlock_data (558), unlock_code (557), lock_linear_region (548), lock_data (548), lock_code (547)

Chapter 11

Reference for unit 'gpm'

11.1 Used units

Table 11.1: Used units by unit 'gpm'

Name	Page
baseUnix	559

11.2 Overview

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

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

11.3 Constants, types and variables

11.3.1 Constants

```
GPM_BOT = 2
```

Bottom of area.

```
GPM_B_LEFT = 4
```

Left mouse button identifier.

```
GPM_B_MIDDLE = 2
```

Middle mouse button identifier.

```
GPM_B_RIGHT = 1
```

Right mouse button identifier.

```
GPM_DOUBLE = 32
```

Mouse double click event.

```
GPM_DOWN = 4
```

Mouse button down event.

```
GPM_DRAG = 2
```

Mouse drag event.

```
GPM_ENTER = 512
```

Enter area event.

```
GPM_HARD = 256
```

?

```
GPM_LEAVE = 1024
```

Leave area event.

```
GPM_LFT = 4
```

Left side of area.

```
GPM_MAGIC = $47706D4C
```

Constant identifying GPM in gpm_Open (567).

```
GPM_MFLAG = 128
```

Motion flag.

```
GPM_MOVE = 1
```

Mouse move event.

```
GPM_NODE_CTL = GPM_NODE_DEV
```

Control socket

```
GPM_NODE_DEV = '/dev/gpmctl'
```

Device socket filename

```
GPM_NODE_DIR = _PATH_VARRUN
```

Where to write socket.

```
GPM_NODE_DIR_MODE = 0775
```

Mode of socket.

```
GPM_NODE_FIFO = '/dev/gpmdata'
```

FIFO name

```
GPM_NODE_PID = '/var/run/gpm.pid'
```

Name of PID file.

```
GPM_RGT = 8
```

Right side of area.

```
GPM_SINGLE = 16
```

Mouse single click event.

```
GPM_TOP = 1
```

Top of area.

```
GPM_TRIPLE = 64
```

Mouse triple click event.

```
GPM_UP = 8
```

Mouse button up event.

```
_PATH_DEV = '/dev/'
```

Location of /dev directory.

```
_PATH_VARRUN = '/var/run/'
```

Location of run PID files directory.

11.3.2 Types

```
Pgpmconnect = Pgpm_connect
```

Pointer to TGpmConnect (563) record.

```
Pgpmevent = Pgpm_event
```

Pointer to TGpmEvent (563) record

```
Pgpmroi = Pgpm_roi
```

Pointer to TGpmRoi (563) record.

```
Pgpm_connect = ^TGpm_connect
```

Pointer to TGpm_Connect (563) record.

```
Pgpm_event = ^Tgpm_event
```

Pointer to TGpm_Event (563) record

```
Pgpm_roi = ^Tgpm_roi
```

Pointer to Tgpm_roi (563) record.

```
Tgpmconnect = Tgpm_connect
```

Alias for TGpm_Connect (563) record.

```
TGpmEtype = LongInt
```

Type for event type.

```
Tgpmevent = Tgpm_event
```

Alias for TGPM_EVent (563) record

```
TGpmHandler = function(var event: Tgpmevent;clientdata: pointer)
                       : LongInt
```

Mouse event handler callback.

```
TGpmMargin = LongInt
```

Type to hold area margin.

```
Tgpmroi = Tgpm_roi
```

Alias for TGpm_roi (563)Record

```
Tgpm_connect = packed record
  eventMask : Word;
  defaultMask : Word;
  minMod : Word;
  maxMod : Word;
  pid : LongInt;
  vc : LongInt;
end
```

GPM server connection information.

```
Tgpm_event = packed record
  buttons : Byte;
  modifiers : Byte;
  vc : Word;
  dx : Word;
  dy : Word;
  x : Word;
  y : Word;
  EventType : TGpmEtype;
  clicks : LongInt;
  margin : TGpmMargin;
  wdx : Word;
  wdy : Word;
end
```

`Tgpm_event` describes the events that are reported by GPM.

```
Tgpm_roi = packed record
  xmin : Integer;
  xmax : Integer;
  ymin : Integer;
  ymax : Integer;
  minmod : Word;
  maxmod : Word;
  eventmask : Word;
  owned : Word;
  handler : TGpmHandler;
  clientdata : pointer;
  prev : Pgpm_roi;
  next : Pgpm_roi;
end
```

Record used to define regions of interest.

11.3.3 Variables

`gpm_current_roi : Pgpm_roi`

Internal gpm library variable. Do not use.

`gpm_handler : TGpmHandler`

Internal gpm library variable. Do not use.

`gpm_roi : Pgpm_roi`

Internal gpm library variable. Do not use.

`gpm_roi_data : pointer`

Internal gpm library variable. Do not use.

```
gpm_roi_handler : TGpmHandler
```

Internal gpm library variable. Do not use.

11.4 Procedures and functions

11.4.1 Gpm_AnyDouble

Synopsis: Check whether event has double click event.

Declaration: `function Gpm_AnyDouble(EventType: LongInt) : Boolean`

Visibility: default

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

Errors: None.

See also: Gpm_StrictSingle (569), Gpm_AnySingle (564), Gpm_StrictDouble (569), Gpm_StrictTriple (569), Gpm_AnyTriple (564)

11.4.2 Gpm_AnySingle

Synopsis: Check whether event has a single click event.

Declaration: `function Gpm_AnySingle(EventType: LongInt) : Boolean`

Visibility: default

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

Errors: None.

See also: Gpm_StrictSingle (569), Gpm_AnyDouble (564), Gpm_StrictDouble (569), Gpm_StrictTriple (569), Gpm_AnyTriple (564)

11.4.3 Gpm_AnyTriple

Synopsis: Check whether event has a triple click event.

Declaration: `function Gpm_AnyTriple(EventType: LongInt) : Boolean`

Visibility: default

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

Errors: None.

See also: Gpm_StrictSingle (569), Gpm_AnyDouble (564), Gpm_StrictDouble (569), Gpm_StrictTriple (569), Gpm_AnySingle (564)

11.4.4 gpm_close

Synopsis: Close connection to GPM server.

Declaration:
```
function gpm_close : LongInt
```

Visibility: default

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

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

for an example, see Gpm_GetEvent (565).

Errors: None.

See also: Gpm_Open (567)

11.4.5 gpm_fitvalues

Synopsis: Change coordinates to fit physical screen.

Declaration:
```
function gpm_fitvalues(var x: LongInt;var y: LongInt) : LongInt
```

Visibility: default

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

Errors: None.

See also: Gpm_FitValuesM (565)

11.4.6 gpm_fitvaluesM

Synopsis: Change coordinates to fit margin.

Declaration:
```
function gpm_fitvaluesM(var x: LongInt;var y: LongInt;margin: LongInt)
                                : LongInt
```

Visibility: default

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

Errors: None.

See also: Gpm_FitValues (565)

11.4.7 gpm_getevent

Synopsis: Get event from event queue.

Declaration:
```
function gpm_getevent(var event: Tgpm_event) : LongInt
```

Visibility: default

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

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

Errors: On error, -1 is returned.

See also: Gpm_GetSnapshot (567)

Listing: ./gpmex/gpmex.pp

```
program gpmex;

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

uses gpm;

var
  connect : TGPMConnect;
  event : tgpmevent;

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

11.4.8 gpm_getsnapshot

Synopsis: Return servers' current image of mouse state.

Declaration:
```
function gpm_getsnapshot(eptr: Pgpmevent) : LongInt
function gpm_getsnapshot(var eptr: Tgpmevent) : LongInt
```

Visibility: default

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

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

Errors: None.

See also: Gpm_GetEvent (565)

11.4.9 gpm_lowerroi

Synopsis: Lower a region of interest in the stack.

Declaration:
```
function gpm_lowerroi(which: Pgpm_roi;after: Pgpm_roi) : Pgpm_roi
```

Visibility: default

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

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

Errors: None.

See also: Gpm_RaiseRoi (568), Gpm_PopRoi (568), Gpm_PushRoi (568)

11.4.10 gpm_open

Synopsis: Open connection to GPM server.

Declaration:
```
function gpm_open(var conn: Tgpm_connect;flag: LongInt) : LongInt
```

Visibility: default

Description: `Gpm_Open` opens a new connection to the mouse server. The connection is described by the fields of the `conn` record of type TGPMConnect (563).

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

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

for an example, see Gpm_GetEvent (565).

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

See also: Gpm_Open (567)

11.4.11 gpm_poproi

Synopsis: Pop region of interest from the stack.

Declaration:
```
function gpm_poproi(which: Pgpm_roi) : Pgpm_roi
```

Visibility: default

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

Errors: None.

See also: Gpm_RaiseRoi (568), Gpm_LowerRoi (567), Gpm_PushRoi (568)

11.4.12 gpm_pushroi

Synopsis: Push region of interest on the stack.

Declaration:
```
function gpm_pushroi(x1: LongInt;y1: LongInt;x2: LongInt;y2: LongInt;
                     mask: LongInt;fun: TGpmHandler;xtradata: pointer)
                      : Pgpm_roi
```

Visibility: default

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

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

Errors: None.

See also: Gpm_RaiseRoi (568), Gpm_PopRoi (568), Gpm_LowerRoi (567)

11.4.13 gpm_raiseroi

Synopsis: Raise region of interest in the stack.

Declaration:
```
function gpm_raiseroi(which: Pgpm_roi;before: Pgpm_roi) : Pgpm_roi
```

Visibility: default

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

Errors: None.

See also: Gpm_PushRoi (568), Gpm_PopRoi (568), Gpm_LowerRoi (567)

11.4.14 gpm_repeat

Synopsis: Check for presence of mouse event.

Declaration:
```
function gpm_repeat(millisec: LongInt) : LongInt
```

Visibility: default

Description: `Gpm_Repeat` returns 1 of no mouse event arrives in the next `millisec` miiliseconds, it returns 0 otherwise.

Errors: None.

See also: Gpm_GetEvent (565)

11.4.15 Gpm_StrictDouble

Synopsis: Check whether event contains only a double-click event.

Declaration: `function Gpm_StrictDouble(EventType: LongInt) : Boolean`

Visibility: default

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

Errors: None.

See also: Gpm_StrictSingle (569), Gpm_AnyTriple (564), Gpm_AnyDouble (564), Gpm_StrictTriple (569), Gpm_AnySingle (564)

11.4.16 Gpm_StrictSingle

Synopsis: Check whether event contains only a single-click event.

Declaration: `function Gpm_StrictSingle(EventType: LongInt) : Boolean`

Visibility: default

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

Errors: None.

See also: Gpm_AnyTriple (564), Gpm_StrictDouble (569), Gpm_AnyDouble (564), Gpm_StrictTriple (569), Gpm_AnySingle (564)

11.4.17 Gpm_StrictTriple

Synopsis: Check whether event contains only a triple-click event.

Declaration: `function Gpm_StrictTriple(EventType: LongInt) : Boolean`

Visibility: default

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

Errors: None.

See also: Gpm_AnyTriple (564), Gpm_StrictDouble (569), Gpm_AnyDouble (564), Gpm_StrictSingle (569), Gpm_AnySingle (564)

Chapter 12

Reference for unit 'Graph'

12.1 Categorized functions: Text and font handling

Functions to set texts on the screen.

Table 12.1:

Name	Description
GetTextSettings (605)	Get current text settings
InstallUserFont (608)	Install a new font
OutText (609)	Write text at current cursor position
OutTextXY (596)	Write text at coordinates X,Y
RegisterBGIFont (611)	Register a new font
SetTextJustify (614)	Set text justification
SetTextStyle (615)	Set text style
SetUserCharSize (615)	Set text size
TextHeight (616)	Calculate height of text
TextWidth (616)	Calculate width of text

12.2 Categorized functions: Filled drawings

Functions for drawing filled regions.

12.3 Categorized functions: Drawing primitives

Functions for simple drawing.

12.4 Categorized functions: Color management

All functions related to color management.

Table 12.2:

Name	Description
Bar3D (598)	Draw a filled 3D-style bar
Bar (598)	Draw a filled rectangle
FloodFill (600)	Fill starting from coordinate
FillEllipse (599)	Draw a filled ellipse
FillPoly (600)	Draw a filled polygone
GetFillPattern (602)	Get current fill pattern
GetFillSettings (602)	Get current fill settings
SetFillPattern (612)	Set current fill pattern
SetFillStyle (613)	Set current fill settings

Table 12.3:

Name	Description
Arc (597)	Draw an arc
Circle (595)	Draw a complete circle
DrawPoly (599)	Draw a polygone with N points
Ellipse (599)	Draw an ellipse
GetArcCoords (600)	Get arc coordinates
GetLineSettings (603)	Get current line drawing settings
Line (596)	Draw line between 2 points
LineRel (608)	Draw line relative to current position
LineTo (609)	Draw line from current position to absolute position
MoveRel (609)	Move cursor relative to current position
MoveTo (609)	Move cursor to absolute position
PieSlice (610)	Draw a pie slice
PutPixel (597)	Draw 1 pixel
Rectangle (610)	Draw a non-filled rectangle
Sector (611)	Draw a sector
SetLineStyle (613)	Set current line drawing style

12.5 Categorized functions: Screen management

General drawing screen management functions.

12.6 Categorized functions: Initialization

Initialization of the graphics screen.

12.7 Target specific issues: Linux

There are several issues on Linux that need to be taken care of:

The Linux version of the Graph unit uses the libvga library. This library works on the console, not under X.

If you get an error similar to

Table 12.4:

Name	Description
GetBkColor (601)	Get current background color
GetColor (601)	Get current foreground color
GetDefaultPalette (601)	Get default palette entries
GetMaxColor (603)	Get maximum valid color
GetPaletteSize (605)	Get size of palette for current mode
GetPixel (596)	Get color of selected pixel
GetPalette (605)	Get palette entry
SetAllPalette (597)	Set all colors in palette
SetBkColor (612)	Set background color
SetColor (612)	Set foreground color
SetPalette (614)	Set palette entry
SetRGBPalette (597)	Set palette entry with RGB values

Table 12.5:

Name	Description
ClearViewPort (595)	Clear the current viewport
GetImage (595)	Copy image from screen to memory
GetMaxX (604)	Get maximum X coordinate
GetMaxY (604)	Get maximum Y coordinate
GetX (606)	Get current X position
GetY (606)	Get current Y position
ImageSize (596)	Get size of selected image
GetViewSettings (605)	Get current viewport settings
PutImage (596)	Copy image from memory to screen
SetActivePage (597)	Set active video page
SetAspectRatio (611)	Set aspect ratio for drawing routines
SetViewPort (616)	Set current viewport
SetVisualPage (597)	Set visual page
SetWriteMode (616)	Set write mode for screen operations

```
/usr/bin/ld: cannot find -lvga
```

This can mean one of two things: either libvga and it's development package is not installed properly, or the directory where it is installed is not in the linker path.

To remedy the former, you should install both the libvga package and libvga-devel package (or compile and install from scratch).

To remedy the latter, you should add the path to the compiler command-line using the `-Fl` option.

Programs using libvga need root privileges to run. You can make them setuid root with the following command:

```
chown root.root myprogram
chmod u+s myprogram
```

The libvga library will give up the root privileges after it is initialized.

there is an experimental version of the Graphics library available that uses GGI to do all the drawing, but it is not well tested. It's called ggigraph and is distributed in source form only.

Table 12.6:

Name	Description
ClearDevice (598)	Empty the graphics screen
CloseGraph (598)	Finish drawing session, return to text mode
DetectGraph (599)	Detect graphical modes
GetAspectRatio (601)	Get aspect ratio of screen
GetModeRange (604)	Get range of valid modes for current driver
GraphDefaults (606)	Set defaults
GetDriverName (602)	Return name of graphical driver
GetGraphMode (603)	Return current or last used graphics mode
GetMaxMode (603)	Get maximum mode for current driver
GetModeName (604)	Get name of current mode
GraphErrorMsg (606)	String representation of graphical error
GraphResult (607)	Result of last drawing operation
InitGraph (607)	Initialize graphics drivers
InstallUserDriver (608)	Install a new driver
RegisterBGIDriver (610)	Register a new driver
RestoreCRTMode (611)	Go back to text mode
SetGraphMode (613)	Set graphical mode

Do not use the CRT unit together with the Graph unit: the console may end up in an unusable state. Instead, the ncurses unit may function fine.

12.8 Target specific issues: DOS

VESA modes (i.e., anything but 320x200x256 and 640x480x16) do not work under most installations of Windows NT, Windows 2000 and Windows XP. They also do not work for some people under Windows 98 and Windows ME, depending on their graphics drivers. However, the graph unit cannot detect this, because no errors are returned from the system. In such cases, the screen simply turns black, or will show garbage.

Nothing can be done about this, the reason is missing or buggy support in the graphics drivers of the operating system.

12.9 A word about mode selection

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

However, since

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

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

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

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

lowNewDriver = 11;
highNewDriver = 21;
```

Each of these drivers specifies a desired color-depth.

The following modes have been defined:

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

lowNewMode = 30001;
highNewMode = 30015;
```

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

The above constants can be used to set a certain color depth and resultion, as demonstrated in the below example.

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

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

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

12.10 Requirements

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

12.11 Overview

This document describes the GRAPH unit for Free Pascal, for all platforms. The unit was first written for dos by Florian klaempfl, but was later completely rewritten by Carl-Eric Codere to be completely portable. The unit is provided for compatibility only: It is recommended to use more modern graphical systems. The graph unit will allow to recompile old programs, they will work to some extent, but if the application has heavy graphical needs, it's recommended to use another set of graphical routines, suited to the platform the program should work on.

12.12 Constants, types and variables

12.12.1 Constants

```
AndPut = 3
```

Draw operation: use AND

```
AnsiToASCIITransTable : TCharsetTransTable = (#$00,#$01,#$02,#$03,#$04,#$05,#$06,#$0
```

Default ansi transliteration table.

```
BkSlashFill = 5
```

Fill style: Diagonal (backslash) lines

```
black = 0
```

Color code: black.

```
blue = 1
```

Color code: blue

```
BoldFont = 10
```

Font number: Bold font.

```
BottomText = 0
```

Vertical text alignment: Align text to bottom

```
brown = 6
```

Color code: brown

```
CenterLn = 2
```

Line style: centered line

```
CenterText = 1
```

Horizontal text alignment: Center text

```
ClipOff = false
```

Viewport clipping off

```
ClipOn = true
```

Viewport clipping on

```
CloseDotFill = 11
```

Fill style: Closely spaced dotted lines

```
CopyPut = 0
```

Draw operation: use Copy

```
CurrentDriver = -128
```

Currently used driver

```
cyan = 3
```

Color code: Cyan

```
D12bit = 16
```

Mode: Depth 12 bit

```
D15bit = 17
```

Mode: Depth 15 bit

```
D16bit = 18
```

Mode: Depth 16 bit

```
D1bit = 11
```

Mode: Depth 1 bit

```
D24bit = 19
```

Mode: Depth 24 bit

```
D2bit = 12
```

Mode: Depth 2 bit

```
D32bit = 20
```

Mode: Depth 32 bit

```
D4bit = 13
```

Mode: Depth 4 bit

```
D64bit = 21
```

Mode: Depth 64 bit

```
D6bit = 14
```

Mode: Depth 6 bit

```
D8bit = 15
```

Mode: Depth 8 bit

```
darkgray = 8
```

Color code: Dark gray

```
DashedLn = 3
```

Line style: dashed line

```
Default = 0
```

Default mode

```
DefaultFont = 0
```

Font number: Normal font

```
Detect = 0
```

Mode: Detect mode.

```
detectMode = 30000
```

Mode: Autodetect optimal mode

```
DottedLn = 1
```

Line style: Dotted line

```
DrawTextBackground : Boolean = false
```

Should the background of texts be drawn or should it be left untouched ?

```
EGABlack = 0
```

Color code: EGA Black

```
EGABlue = 1
```

Color code: EGA blue

```
EGABrown = 20
```

Color code: EGA brown

```
EGACyan = 3
```

Color code: EGA cyan

```
EGADarkgray = 56
```

Color code: EGA dark gray

```
EGAGreen = 2
```

Color code: EGA green

```
EGALightblue = 57
```

Color code: EGA Light blue

```
EGALightcyan = 59
```

Color code: EGA Light cyan

```
EGALightgray = 7
```

Color code: EGA Light gray

```
EGALightgreen = 58
```

Color code: EGA Light green

```
EGALightmagenta = 61
```

Color code: EGA light magenta

```
EGALightred = 60
```

Color code: EGA light red

```
EGAMagenta = 5
```

Color code: EGA magenta

```
EGARed = 4
```

Color code: EGA red

```
EGAWhite = 63
```

Color code: EGA white

```
EGAYellow = 62
```

Color code: EGA yellow

```
EmptyFill = 0
```

Fill style: Do not fill

```
EuroFont = 9
```

Font number: ?

```
fillpatternTable : Array[0..12] of FillPatternType = ( ( $00,$00,$00,$00,$00,$00,$00
```

Table with standard fill patterns

```
G1024x768x16 = 30
```

Mode: Resolution 1024x768, 16 colors

```
G1024x768x16M = 25
```

Mode: Resolution 1024x768, 16M colors

```
G1024x768x16M32 = 36
```

Mode: Resolution 1024x758, 16M 32-bit colors

```
G1024x768x256 = 12
```

Mode: Resolution 1024x768, 256 colors

```
G1024x768x32K = 23
```

Mode: Resolution 1024x768, 32K colors

```
G1024x768x64K = 24
```

Mode: Resolution 1024x768, 64K colors

```
G1152x864x16 = 38
```

Mode: Resolution 1152x864, 16 colors

```
G1152x864x16M = 42
```

Mode: Resolution 1152x864, 16M colors

```
G1152x864x16M32 = 43
```

Mode: Resolution 1152x864, 16M 32-bitcolors

```
G1152x864x256 = 39
```

Mode: Resolution 1152x864, 256 colors

```
G1152x864x32K = 40
```

Mode: Resolution 1152x864, 32K colors

```
G1152x864x64K = 41
```

Mode: Resolution 1152x864, 64K colors

```
G1280x1024x16 = 31
```

Mode: Resolution 1280x1024, 16 colors

```
G1280x1024x16M = 28
```

Mode: Resolution 1280x1024, 16M colors

```
G1280x1024x16M32 = 37
```

Mode: Resolution 1280x1024, 16M 32-bit colors

```
G1280x1024x256 = 13
```

Mode: Resolution 1280x1024, 256 colors

```
G1280x1024x32K = 26
```

Mode: Resolution 1280x1024, 32K colors

```
G1280x1024x64K = 27
```

Mode: Resolution 1280x1024, 64K colors

```
G1600x1200x16 = 44
```

Mode: Resolution 1600x1200, 16 colors

```
G1600x1200x16M = 48
```

Mode: Resolution 1600x1200, 16M colors

```
G1600x1200x16M32 = 49
```

Mode: Resolution 1600x1200, 16M 32-bit colors

```
G1600x1200x256 = 45
```

Mode: Resolution 1600x1200, 256 colors

```
G1600x1200x32K = 46
```

Mode: Resolution 1600x1200, 32K colors

```
G1600x1200x64K = 47
```

Mode: Resolution 1600x1200, 64K colors

```
G320x200x16 = 1
```

Mode: Resulution 320x200, 16 colors

```
G320x200x16M = 16
```

Mode: Resolution 320x200, 16M colors

```
G320x200x16M32 = 33
```

Mode: Resolution 320x200, 16M 32-bit colors

```
G320x200x256 = 5
```

Mode: Resolution 320x200, 256 colors

```
G320x200x32K = 14
```

Mode: Resolution 320x200, 32K colors

```
G320x200x64K = 15
```

Mode: Resolution 320x200, 64K colors

```
G320x240x256 = 6
```

Mode: Resolution 320x240, 256 colors

```
G320x400x256 = 7
```

Mode: Resolution 320x400, 256 colors

```
G360x480x256 = 8
```

Mode: Resolution 360x480, 256 colors

```
G640x200x16 = 2
```

Mode: Resolution x, colors

```
G640x350x16 = 3
```

Mode: Resolution x, colors

```
G640x480x16 = 4
```

Mode: Resolution x, colors

```
G640x480x16M = 19
```

Mode: Resolution 640x480, 16M colors

```
G640x480x16M32 = 34
```

Mode: Resolution 640x480, 16M 32-bit colors

```
G640x480x2 = 9
```

Mode: Resolution 640x480, 2 colors

```
G640x480x256 = 10
```

Mode: Resolution 640x480, 256 colors

```
G640x480x32K = 17
```

Mode: Resolution 640x480, 32K colors

```
G640x480x64K = 18
```

Mode: Resolution 640x480, 64K colors

```
G720x348x2 = 32
```

Mode: Resolution 720x348, 2 colors

```
G800x600x16 = 29
```

Mode: Resolution 800x600, 16 colors

```
G800x600x16M = 22
```

Mode: Resolution 800x600, 16M colors

```
G800x600x16M32 = 35
```

Mode: Resolution 800x600, 16M 32-bit colors

```
G800x600x256 = 11
```

Mode: Resolution 800x600, 256 colors

```
G800x600x32K = 20
```

Mode: Resolution 800x600, 32K colors

```
G800x600x64K = 21
```

Mode: Resolution 800x600, 64K colors

```
GothicFont = 4
```

Font number: Gothic font

```
GraphStringTransTable : PCharsetTransTable = nil
```

Table used when transliterating strings.

```
green = 2
```

Color code: green

```
grError = -11
```

Error: Unknown error.

```
grFileNotFound = -3
```

Error: File for driver not found.

```
grFontNotFound = -8
```

Error: font description file not found.

```
grInvalidDriver = -4
```

Error: Invalid driver specified

```
grInvalidFont = -13
```

Error: Invalid font description

```
grInvalidFontNum = -14
```

Error: Invalid font number

```
grInvalidMode = -10
```

Error: Invalid mode specified.

```
grInvalidVersion = -18
```

Error: Invalid version.

```
grIOerror = -12
```

Error: Unspecified Input/Output error.

```
grNoFloodMem = -7
```

Error: Could not allocate memory for flood operation.

```
grNoFontMem = -9
```

Error: Not enough memory to load font.

```
grNoInitGraph = -1
```

Error: Graphical system not initialized

```
grNoLoadMem = -5
```

Error: Memory error.

```
grNoScanMem = -6
```

Error: Could not allocate memory for scan

```
grNotDetected = -2
```

Error: Graphics device not detected.

```
grOk = 0
```

Graphical operation went OK.

```
HatchFill = 7
```

Fill style: Hatch lines

```
HercMono = 7
```

Mode: Hercules, mono color

```
HercMonoHi = 0
```

Mode: Hercules card, monochrome, high resolution

```
highNewDriver = 21
```

Mode: highest number for new driver

```
highNewMode = 30015
```

Mode: Highest possible value of the new modes.

```
HorizDir = 0
```

Text write direction: Horizontal

```
InterleaveFill = 9
```

Fill style: Interleaving lines

```
LCOMFont = 8
```

Font number: ?

```
LeftText = 0
```

Horizontal text alignment: Align text left

```
lightblue = 9
```

Color code: Light blue

```
lightcyan = 11
```

Color code: Light cyan

```
lightgray = 7
```

Color code: Light gray

```
lightgreen = 10
```

Color code: Light green

```
lightmagenta = 13
```

Color code: Light magenta

```
lightred = 12
```

Color code: Light red

```
LineFill = 2
```

Fill style: Fill using horizontal lines

```
lowNewDriver = 11
```

Mode: lowest number for new driver

```
lowNewMode = 30001
```

Mode: Lowest possible value of the new modes.

```
LowRes = 1
```

Mode: Low resolution.

```
LtBkSlashFill = 6
```

Fill style: Light diagonal (backslash) lines

```
LtSlashFill = 3
```

Fill style: Light diagonal (slash) lines

```
m1024x768 = 30012
```

Mode: Resolution 1024x768

```
m1280x1024 = 30013
```

Mode: Resolution 1280x1024

```
m1600x1200 = 30014
```

Mode: Resolution 1600x1200

```
m2048x1536 = 30015
```

Mode: Resolution 2048x1536

```
m320x200 = 30001
```

Mode: Resolution 320x200

```
m320x256 = 30002
```

Mode: Resolution 320x256

```
m320x400 = 30003
```

Mode: Resolution 320x400

```
m512x384 = 30004
```

Mode: Resolution 512x384

```
m640x200 = 30005
```

Mode: Resolution 640x200

```
m640x256 = 30006
```

Mode: Resolution 640x256

```
m640x350 = 30007
```

Mode: Resolution 640x350

```
m640x400 = 30008
```

Mode: Resolution 640x400

```
m640x480 = 30009
```

Mode: Resolution 640x480

```
m800x600 = 30010
```

Mode: Resolution 800x600

```
m832x624 = 30011
```

Mode: Resolution 832x624

```
magenta = 5
```

Color code: Magenta

```
MaxColors = 255
```

Max amount of colors in a palette

```
maxsmallint = high ( smallint )
```

Maximum value for smallint type

```
NormalPut = 0
```

Draw operation: Use Normal (copy) operation

```
NormWidth = 1
```

Line width: Normal width

```
NotPut = 4
```

Draw operation: use NOT

```
OrPut = 2
```

Draw operation: use OR

```
red = 4
```

Color code: Red

```
resolutions : Array[lowNewMode..highNewMode] of TResolutionRec = ( ( x:320;y:200 ) ,
```

Array with actual resolutions of the new modes

```
RightText = 2
```

Horizontal text alignment: Align text right

```
SansSerifFont = 3
```

Font number: Sans Serif font

```
ScriptFont = 5
```

Font number: Script font

```
SimpleFont = 6
```

Font number: Simple font

```
SlashFill = 4
```

Fill style: Diagonal (slash) lines

```
SmallFont = 2
```

Font number: Small font

```
SolidFill = 1
```

Fill style: Solid fill.

```
SolidLn = 0
```

Line style: Solid line

```
ThickWidth = 3
```

Line width: double width

```
TopOff = false
```

Top off

```
TopOn = true
```

Top on

```
TopText = 2
```

Vertical text alignment: Align text to top

```
TriplexFont = 1
```

Font number: Triplex font

```
TSCRFont = 7
```

Font number: Terminal font

```
UserBitLn = 4
```

Line style: User defined

```
UserCharSize = 0
```

User character size

```
UserFill = 12
```

Fill style: User-defined fill.

```
VertDir = 1
```

Text write direction: Vertical

```
VESA = 10
```

Mode: VESA graphics adaptor.

```
VGA = 9
```

Mode: VGA graphics adaptor.

```
VGAHi = 2
```

Mode: VGA high resolution (640x480)

```
VGALo = 0
```

Mode: VGA low resolution (640x200)

```
VGAMed = 1
```

Mode: VGA medium resolution (640x350)

```
white = 15
```

Color code: White

```
WideDotFill = 10
```

Fill style: Widely spaced dotted lines

```
XHatchFill = 8
```

Fill style: Heavy hatch lines

```
XORPut = 1
```

Draw operation: use XOR

```
yellow = 14
```

Color code: Yellow

12.12.2 Types

```
ArcCoordsType = record
  x : SmallInt;
  y : SmallInt;
  xstart : SmallInt;
  ystart : SmallInt;
  xend : SmallInt;
  yend : SmallInt;
end
```

Describe the last arc which was drawn on screen

```
CircleProc = procedure(X: SmallInt;Y: SmallInt;Radius: Word)
```

Standard circle drawing routine prototype.

```
clrviewproc = procedure
```

Standard clearviewport routine prototype

```
defpixelproc = procedure(X: SmallInt;Y: SmallInt)
```

This is the standard putpixel routine used by all function drawing routines, it will use the viewport settings, as well as clip, and use the current foreground color to plot the desired pixel.

```
ellipseproc = procedure(X: SmallInt;Y: SmallInt;XRadius: Word;
                        YRadius: Word;stAngle: Word;EndAngle: Word;
                        fp: patternlineproc)
```

Standard ellipse drawing routine prototype.

```
FillPatternType = Array[1..8] of Byte
```

Bit pattern used when drawing lines. Set bits are drawn.

```
FillSettingsType = record
  pattern : Word;
  color : Word;
end
```

Record describing fill mode

```
getimageproc = procedure(X1: SmallInt;Y1: SmallInt;X2: SmallInt;
                         Y2: SmallInt;var Bitmap)
```

Standard GetImage (595) procedure prototype.

```
getpixelproc = function(X: SmallInt;Y: SmallInt) : Word
```

Standard pixel fetching routine prototype

```
getrgbpaletteproc = procedure(ColorNum: SmallInt;var RedValue: SmallInt;
                              var GreenValue: SmallInt;
                              var BlueValue: SmallInt)
```

This routine prototype is a hook for GetRGBPalette (596)

```
getscanlineproc = procedure(X1: SmallInt;X2: SmallInt;Y: SmallInt;
                            var data)
```

This routine is used for FloodFill (600) It returns an entire screen scan line with a word for each pixel in the scanline. Also handy for GetImage.

```
graphfreememprc = procedure(var P: Pointer;size: Word)
```

Procedure prototype, used when heap memory is freed by the graph routines.

```
graphgetmemprc = procedure(var P: pointer;size: Word)
```

Procedure prototype, used when heap memory is needed by the graph routines.

```
graph_float = single
```

The platform's preferred floating point size for fast graph operations

```
hlineproc = procedure(x: SmallInt;x2: SmallInt;y: SmallInt)
```

Standard procedure prototype to draw a single horizontal line

```
imagesizeproc = function(X1: SmallInt;Y1: SmallInt;X2: SmallInt;
                         Y2: SmallInt) : LongInt
```

Standard ImageSize (596) calculation procedure prototype.

```
initmodeproc = procedure
```

Standard routine prototype to initialize a mode.

```
lineproc = procedure(X1: SmallInt;Y1: SmallInt;X2: SmallInt;
                     Y2: SmallInt)
```

Standard line drawing routine prototype.

```
LineSettingsType = record
  linestyle : Word;
  pattern : Word;
  thickness : Word;
end
```

Record describing current line drawing mode

```
OutTextXYProc = procedure(x: SmallInt;y: SmallInt;
                          const TextString: String)
```

This routine prototype is a hook for OutTextXY (596)

```
PaletteType = record
  Size : LongInt;
  Colors : Array[0..MaxColors] of RGBRec;
end
```

Record describing palette.

```
patternlineproc = procedure(x1: SmallInt;x2: SmallInt;y: SmallInt)
```

Standard procedure prototype to draw a patterned line

```
PCharsetTransTable = ^TCharsetTransTable
```

Pointer to TCharsetTransTable (594) array.

```
PModeInfo = ^TModeInfo
```

Pointer to TModeInfo (594) record

```
PointType = record
  x : SmallInt;
  y : SmallInt;
end
```

Record describing a point in a 2 dimensional plane

```
putimageproc = procedure(X: SmallInt;Y: SmallInt;var Bitmap;
                         BitBlt: Word)
```

Standard PutImage (596) procedure prototype.

```
putpixelproc = procedure(X: SmallInt;Y: SmallInt;Color: Word)
```

Standard pixel drawing routine prototype

```
restorestateproc = procedure
```

Standard routine prototype to restore the graphical state at a closegraph call.

```
RGBRec = packed record
  Red : SmallInt;
  Green : SmallInt;
  Blue : SmallInt;
end
```

Record describing palette RGB color

```
savestateproc = procedure
```

Standard routine prototype to save the graphical state before a mode is set.

```
setactivepageproc = procedure(page: Word)
```

Standard SetActivePage (597) procedure prototype.

```
SetAllPaletteProc = procedure(const Palette: PaletteType)
```

This routine prototype is a hook for SetAllPalette (597)

```
setrgbpaletteproc = procedure(ColorNum: SmallInt;RedValue: SmallInt;
                              GreenValue: SmallInt;BlueValue: SmallInt)
```

This routine prototype is a hook for SetRGBPalette (597)

```
setvisualpageproc = procedure(page: Word)
```

Standard SetVisualPage (597) procedure prototype.

```
smallint = -32768..32767
```

Type redefinition

```
TCharsetTransTable = Array[Char] of Char
```

Character transliteration table, with entries for 256 characters

```
TextSettingsType = record
  font : Word;
  direction : Word;
  charsize : Word;
  horiz : Word;
  vert : Word;
end
```

Record describing how texts are drawn.

```
TModeInfo = record
  DriverNumber : SmallInt;
  ModeNumber : SmallInt;
  internModeNumber : SmallInt;
  MaxColor : LongInt;
  PaletteSize : LongInt;
  XAspect : Word;
  YAspect : Word;
  MaxX : Word;
  MaxY : Word;
  DirectColor : Boolean;
  Hardwarepages : Byte;
  ModeName : String;
  DirectPutPixel : defpixelproc;
  GetPixel : getpixelproc;
  PutPixel : putpixelproc;
  SetRGBPalette : setrgbpaletteproc;
  GetRGBPalette : getrgbpaletteproc;
  SetAllPalette : SetAllPaletteProc;
  SetVisualPage : setvisualpageproc;
  SetActivePage : setactivepageproc;
  ClearViewPort : clrviewproc;
  PutImage : putimageproc;
  GetImage : getimageproc;
  ImageSize : imagesizeproc;
  GetScanLine : getscanlineproc;
  Line : lineproc;
  InternalEllipse : ellipseproc;
  PatternLine : patternlineproc;
  HLine : hlineproc;
  VLine : vlineproc;
  Circle : CircleProc;
  InitMode : initmodeproc;
  OutTextXY : OutTextXYProc;
  next : PModeInfo;
end
```

Record describing a graphical mode.

```
TNewModeInfo = record
```

```
  modeInfo : Array[lowNewDriver..highNewDriver] of PModeInfo;
  loHiModeNr : Array[lowNewDriver..highNewDriver] of ;
end
```

Mode information for new modes.a

```
TResolutionRec = record
  x : LongInt;
  y : LongInt;
end
```

Record describing resolution

```
ViewPortType = record
  x1 : SmallInt;
  y1 : SmallInt;
  x2 : SmallInt;
  y2 : SmallInt;
  Clip : Boolean;
end
```

Record describing a viewport

```
vlineproc = procedure(x: SmallInt;y: SmallInt;y2: SmallInt)
```

Standard procedure prototype to draw a single vertical line

12.12.3 Variables

`Circle : CircleProc`

`Circle` draws a complete circle with center at `(X,Y)`, radius `radius`.

`ClearViewPort : clrviewproc`

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

`DirectPutPixel : defpixelproc`

Hook to directly draw a pixel on the screen.

`GetImage : getimageproc`

`GetImage` Places a copy of the screen area `(X1,Y1)` to `X2,Y2` in `BitMap`

`GetPixel : getpixelproc`

GetPixel returns the color of the point at (X,Y)

GetRGBPalette : getrgbpaletteproc

Hook to set a RGB palette entries.

GetScanLine : getscanlineproc

Hook to get a scan line from the screen.

GraphFreeMemPtr : graphfreememprc

Hook to free heap memory.

GraphGetMemPtr : graphgetmemprc

Hook to get heap memory

HLine : hlineproc

Hook to draw a solid horizontal line

ImageSize : imagesizeproc

ImageSize returns the number of bytes needed to store the image in the rectangle defined by (X1,Y1) and (X2,Y2).

InternalEllipse : ellipseproc

Hook to draw an ellipse

Line : lineproc

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)

OutTextXY : OutTextXYProc

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.

PatternLine : patternlineproc

Hook to draw a patterned line

PutImage : putimageproc

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

`PutPixel : putpixelproc`

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

`RestoreVideoState : restorestateproc`

Hook to restore a saved video mode

`SaveVideoState : savestateproc`

Hook to save the current video state

`SetActivePage : setactivepageproc`

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

`SetAllPalette : SetAllPaletteProc`

Sets the current palette to `Palette`. `Palette` is an untyped variable, usually pointing to a record of type `PaletteType`

`SetRGBPalette : setrgbpaletteproc`

`SetRGBPalette` sets the `ColorNr`-th entry in the palette to the color with RGB-values `Red, Green Blue`.

`SetVisualPage : setvisualpageproc`

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

`VLine : vlineproc`

Hook to draw a solid vertical line

12.13 Procedures and functions

12.13.1 Arc

Synopsis: Draw part of a circle

Declaration:
```
procedure Arc(X: SmallInt;Y: SmallInt;StAngle: Word;EndAngle: Word;
              Radius: Word)
```

Visibility: default

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 (595), Ellipse (599), GetArcCoords (600), PieSlice (610), Sector (611)

12.13.2 Bar

Synopsis: Draw filled rectangle

Declaration:
```
procedure Bar(x1: SmallInt;y1: SmallInt;x2: SmallInt;y2: SmallInt)
```

Visibility: default

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 (598), Rectangle (610)

12.13.3 Bar3D

Synopsis: Draw filled 3-dimensional rectangle

Declaration:
```
procedure Bar3D(x1: SmallInt;y1: SmallInt;x2: SmallInt;y2: SmallInt;
                depth: Word;top: Boolean)
```

Visibility: default

Description: `Bar3d` 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 (598), Rectangle (610)

12.13.4 ClearDevice

Synopsis: Clear the complete screen

Declaration:
```
procedure ClearDevice
```

Visibility: default

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

Errors: None.

See also: ClearViewPort (595), SetBkColor (612)

12.13.5 Closegraph

Synopsis: Close graphical system.

Declaration:
```
procedure Closegraph
```

Visibility: default

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

Errors: None.

See also: InitGraph (607)

12.13.6 DetectGraph

Synopsis: Detect correct graphical driver to use

Declaration:
```
procedure DetectGraph(var GraphDriver: SmallInt;var GraphMode: SmallInt)
```

Visibility: default

Description: `DetectGraph` 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 (607)

12.13.7 DrawPoly

Synopsis: Draw a polygone

Declaration:
```
procedure DrawPoly(NumPoints: Word;var polypoints)
```

Visibility: default

Description: `DrawPoly` draws a polygone with `NumberOfPoints` corner points, using the current color and line-style. PolyPoints is an array of type PointType (593).

Errors: None.

See also: Bar (598), Bar3D (598), Rectangle (610)

12.13.8 Ellipse

Synopsis: Draw an ellipse

Declaration:
```
procedure Ellipse(X: SmallInt;Y: SmallInt;stAngle: Word;EndAngle: Word;
                  XRadius: Word;YRadius: Word)
```

Visibility: default

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

Errors: None.

See also: Arc (597), Circle (595), FillEllipse (599)

12.13.9 FillEllipse

Synopsis: Draw and fill an ellipse

Declaration:
```
procedure FillEllipse(X: SmallInt;Y: SmallInt;XRadius: Word;
                      YRadius: Word)
```

Visibility: default

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 (597), Circle (595), GetArcCoords (600), PieSlice (610), Sector (611)

12.13.10 FillPoly

Synopsis: Draw, close and fill a polygone

Declaration: `procedure FillPoly(NumPoints: Word;var PolyPoints)`

Visibility: default

Description: `FillPoly` 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 (598), Bar3D (598), Rectangle (610)

12.13.11 FloodFill

Synopsis: Fill an area with a given color

Declaration: `procedure FloodFill(x: SmallInt;y: SmallInt;Border: Word)`

Visibility: default

Description: Fills the area containing the point `(X,Y)`, bounded by the color `BorderColor`.

Errors: None

See also: SetColor (612), SetBkColor (612)

12.13.12 GetArcCoords

Synopsis: Return coordinates of last drawn arc or ellipse.

Declaration: `procedure GetArcCoords(var ArcCoords: ArcCoordsType)`

Visibility: default

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

Errors: None.

See also: Arc (597), Ellipse (599)

12.13.13 GetAspectRatio

Synopsis: Return screen resolution

Declaration: `procedure GetAspectRatio(var Xasp: Word;var Yasp: Word)`

Visibility: default

Description: `GetAspectRatio` determines the effective resolution of the screen. The aspect ration can then be calculated as `Xasp/Yasp`.

Errors: None.

See also: InitGraph (607), SetAspectRatio (611)

12.13.14 GetBkColor

Synopsis: Return current background color

Declaration: `function GetBkColor : Word`

Visibility: default

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

Errors: None.

See also: GetColor (601), SetBkColor (612)

12.13.15 GetColor

Synopsis: Return current drawing color

Declaration: `function GetColor : Word`

Visibility: default

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

Errors: None.

See also: GetColor (601), SetBkColor (612)

12.13.16 GetDefaultPalette

Synopsis: Return default palette

Declaration: `procedure GetDefaultPalette(var Palette: PaletteType)`

Visibility: default

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

Errors: None.

See also: GetColor (601), GetBkColor (601)

12.13.17 GetDirectVideo

Synopsis: Determine whether direct video mode is active.

Declaration: `function GetDirectVideo : Boolean`

Visibility: default

Description: Determine whether direct video mode is active.

Errors:

12.13.18 GetDriverName

Synopsis: Return current driver name

Declaration: `function GetDriverName : String`

Visibility: default

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

Errors: None.

See also: GetModeName (604), InitGraph (607)

12.13.19 GetFillPattern

Synopsis: Return current fill pattern

Declaration: `procedure GetFillPattern(var FillPattern: FillPatternType)`

Visibility: default

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

Errors: None

See also: SetFillPattern (612)

12.13.20 GetFillSettings

Synopsis: Return current fill settings

Declaration: `procedure GetFillSettings(var Fillinfo: FillSettingsType)`

Visibility: default

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

Errors: None.

See also: SetFillPattern (612)

12.13.21 GetGraphMode

Synopsis: Get current graphical modus

Declaration: `function GetGraphMode : SmallInt`

Visibility: default

Description: `GetGraphMode` returns the current graphical modus

Errors: None.

See also: InitGraph (607)

12.13.22 GetLineSettings

Synopsis: Get current line drawing settings

Declaration: `procedure GetLineSettings(var ActiveLineInfo: LineSettingsType)`

Visibility: default

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

Errors: None.

See also: SetLineStyle (613)

12.13.23 GetMaxColor

Synopsis: return maximum number of colors

Declaration: `function GetMaxColor : Word`

Visibility: default

Description: `GetMaxColor` returns the maximum color-number which can be set with `SetColor`. Contrary to Turbo Pascal, this color isn't always guaranteed to be white (for instance in 256+ color modes).

Errors: None.

See also: SetColor (612), GetPaletteSize (605)

12.13.24 GetMaxMode

Synopsis: Return biggest mode for the current driver

Declaration: `function GetMaxMode : SmallInt`

Visibility: default

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

Errors: None.

See also: InitGraph (607)

12.13.25 GetMaxX

Synopsis: Return maximal X coordinate

Declaration: `function GetMaxX : SmallInt`

Visibility: default

Description: `GetMaxX` returns the maximum horizontal screen length

Errors: None.

See also: GetMaxY (604)

12.13.26 GetMaxY

Synopsis: Return maximal Y coordinate

Declaration: `function GetMaxY : SmallInt`

Visibility: default

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

Errors: None.

See also: GetMaxY (604)

12.13.27 GetModeName

Synopsis: Return description a modus

Declaration: `function GetModeName(ModeNumber: SmallInt) : String`

Visibility: default

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

Errors: None.

See also: GetDriverName (602), InitGraph (607)

12.13.28 GetModeRange

Synopsis: Return lowest and highest modus of current driver

Declaration:
```
procedure GetModeRange(GraphDriver: SmallInt;var LoMode: SmallInt;
                       var HiMode: SmallInt)
```

Visibility: default

Description: `GetModeRange` returns the Lowest and Highest modus of the currently installed driver. If no modes are supported for this driver, HiModus will be -1.

Errors: None.

See also: InitGraph (607)

12.13.29 GetPalette

Synopsis: Return current palette

Declaration: `procedure GetPalette(var Palette: PaletteType)`

Visibility: default

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

Errors: None.

See also: GetPaletteSize (605), SetPalette (614)

12.13.30 GetPaletteSize

Synopsis: Return maximal number of entries in current palette

Declaration: `function GetPaletteSize : SmallInt`

Visibility: default

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

Errors: None.

See also: GetPalette (605), SetPalette (614)

12.13.31 GetTextSettings

Synopsis: Return current text style

Declaration: `procedure GetTextSettings(var TextInfo: TextSettingsType)`

Visibility: default

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 (615), SetTextJustify (614)

12.13.32 GetViewSettings

Synopsis: Return current viewport

Declaration: `procedure GetViewSettings(var viewport: ViewPortType)`

Visibility: default

Description: `GetViewSettings` returns the current viewport and clipping settings in `ViewPort`.

Errors: None.

See also: SetViewPort (616)

12.13.33 GetX

Synopsis: Return current cursor X position

Declaration: `function GetX : SmallInt`

Visibility: default

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

Errors: None.

See also: GetY (606)

12.13.34 GetY

Synopsis: Return current cursor Y position

Declaration: `function GetY : SmallInt`

Visibility: default

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

Errors: None.

See also: GetX (606)

12.13.35 GraphDefaults

Synopsis: Reset graphical mode to defaults

Declaration: `procedure GraphDefaults`

Visibility: default

Description: `GraphDefaults` resets all settings for viewport, palette, foreground and background pattern, line-style and pattern, filling style, filling color and pattern, font, text-placement and text size.

Errors: None.

See also: SetViewPort (616), SetFillStyle (613), SetColor (612), SetBkColor (612), SetLineStyle (613)

12.13.36 GraphErrorMsg

Synopsis: Return a description of an error

Declaration: `function GraphErrorMsg(ErrorCode: SmallInt) : String`

Visibility: default

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 (607)

12.13.37 GraphResult

Synopsis: Result of last graphical operation

Declaration: `function GraphResult : SmallInt`

Visibility: default

Description: `GraphResult` returns an error-code for the last graphical operation. If the returned value is zero, all went well. A value different from zero means an error has occurred. besides all operations which draw something on the screen, the following procedures also can produce a `GraphResult` different from zero:

- InstallUserFont (608)
- SetLineStyle (613)
- SetWriteMode (616)
- SetFillStyle (613)
- SetTextJustify (614)
- SetGraphMode (613)
- SetTextStyle (615)

Errors: None.

See also: GraphErrorMsg (606)

12.13.38 InitGraph

Synopsis: Initialize grpahical system

Declaration:
```
procedure InitGraph(var GraphDriver: SmallInt;var GraphMode: SmallInt;
                    const PathToDriver: String)
```

Visibility: default

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. Free Pascal does not offer BGI fonts like Borland, these must be obtained separately.

Example code:

```
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.
```

Errors: None.

See also: Modes (573), DetectGraph (599), CloseGraph (598), GraphResult (607)

12.13.39 InstallUserDriver

Synopsis: Install a user driver

Declaration:
```
function InstallUserDriver(Name: String;AutoDetectPtr: Pointer)
                                  : SmallInt
```

Visibility: default

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 (607), InstallUserFont (608)

12.13.40 InstallUserFont

Synopsis: Install a user-defined font

Declaration:
```
function InstallUserFont(const FontFileName: String) : SmallInt
```

Visibility: default

Description: `InstallUserFont` adds the font in `FontPath` to the list of fonts of the .BGI system.

Errors: None.

See also: InitGraph (607), InstallUserDriver (608)

12.13.41 LineRel

Synopsis: Draw a line starting from current position in given direction

Declaration:
```
procedure LineRel(Dx: SmallInt;Dy: SmallInt)
```

Visibility: default

Description: `LineRel` draws a line starting from the current pointer position to the point `(DX,DY)`, \textbf{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 (596), LineTo (609)

12.13.42 LineTo

Synopsis: Draw a line starting from current position to a given point

Declaration: `procedure LineTo(X: SmallInt;Y: SmallInt)`

Visibility: default

Description: `LineTo` draws a line starting from the current pointer position to the point `(DX,DY`, \textbf{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 (608), Line (596)

12.13.43 MoveRel

Synopsis: Move cursor relative to current position

Declaration: `procedure MoveRel(Dx: SmallInt;Dy: SmallInt)`

Visibility: default

Description: `MoveRel` moves the pointer to the point `(DX,DY)`, relative to the current pointer position

Errors: None.

See also: MoveTo (609)

12.13.44 MoveTo

Synopsis: Move cursor to absolute position.

Declaration: `procedure MoveTo(X: SmallInt;Y: SmallInt)`

Visibility: default

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

Errors: None.

See also: MoveRel (609)

12.13.45 OutText

Synopsis: Write text on the screen at the current location.

Declaration: `procedure OutText(const TextString: String)`

Visibility: default

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 (596)

12.13.46 PieSlice

Synopsis: Draw a pie-slice

Declaration:
```
procedure PieSlice(X: SmallInt;Y: SmallInt;stangle: SmallInt;
                   endAngle: SmallInt;Radius: Word)
```

Visibility: default

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 (597), Circle (595), Sector (611)

12.13.47 queryadapterinfo

Synopsis: Function called to retrieve the current video adapter settings.

Declaration:
```
function queryadapterinfo : PModeInfo
```

Visibility: default

12.13.48 Rectangle

Synopsis: Draw a rectangle on the screen.

Declaration:
```
procedure Rectangle(x1: SmallInt;y1: SmallInt;x2: SmallInt;y2: SmallInt)
```

Visibility: default

Description: Draws a rectangle with corners at `(X1,Y1)` and `(X2,Y2)`, using the current color and style.

Errors: None.

See also: Bar (598), Bar3D (598)

12.13.49 RegisterBGIDriver

Synopsis: Register a new BGI driver.

Declaration:
```
function RegisterBGIDriver(driver: pointer) : SmallInt
```

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserDriver (608), RegisterBGIFont (611)

12.13.50 RegisterBGIfont

Synopsis: Register a new BGI font

Declaration:
```
function RegisterBGIfont(font: pointer) : SmallInt
```

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserFont (608), RegisterBGIDriver (610)

12.13.51 RestoreCrtMode

Synopsis: Restore text screen

Declaration:
```
procedure RestoreCrtMode
```

Visibility: default

Description: Restores the screen modus which was active before the graphical modus was started.
To get back to the graph mode you were last in, you can use `SetGraphMode(GetGraphMode)`

Errors: None.

See also: InitGraph (607)

12.13.52 Sector

Synopsis: Draw and fill a sector of an ellipse

Declaration:
```
procedure Sector(x: SmallInt;y: SmallInt;StAngle: Word;EndAngle: Word;
                 XRadius: Word;YRadius: Word)
```

Visibility: default

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 (597), Circle (595), PieSlice (610)

12.13.53 SetAspectRatio

Synopsis: Set aspect ration of the screen

Declaration:
```
procedure SetAspectRatio(Xasp: Word;Yasp: Word)
```

Visibility: default

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

Errors: None

See also: InitGraph (607), GetAspectRatio (601)

12.13.54 SetBkColor

Synopsis: Set background drawing color

Declaration: `procedure SetBkColor(ColorNum: Word)`

Visibility: default

Description: Sets the background color to `Color`.

Errors: None.

See also: GetBkColor (601), SetColor (612), SetWriteMode (616)

12.13.55 SetColor

Synopsis: Set foreground drawing color

Declaration: `procedure SetColor(Color: Word)`

Visibility: default

Description: Sets the foreground color to `Color`.

Errors: None.

See also: GetColor (601), SetBkColor (612), SetWriteMode (616)

12.13.56 SetDirectVideo

Synopsis: Attempt to enter direct video mode.

Declaration: `procedure SetDirectVideo(DirectAccess: Boolean)`

Visibility: default

Description: `SetDirectVideo` attempts to enter direct video mode. In that mode, everything is drawn straight in the video buffer.

12.13.57 SetFillPattern

Synopsis: Set drawing fill pattern

Declaration: `procedure SetFillPattern(Pattern: FillPatternType;Color: Word)`

Visibility: default

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 (602), SetFillStyle (613), SetWriteMode (616)

12.13.58 SetFillStyle

Synopsis: Set drawing fill style

Declaration: `procedure SetFillStyle(Pattern: Word;Color: Word)`

Visibility: default

Description: `SetFillStyle` sets the filling pattern and color to one of the predefined filling patterns. `Pattern` can be one of the following predefined constants :

EmptyFillUses backgroundcolor.

SolidFillUses filling color

LineFillFills with horizontal lines.

ltSlashFillFills with lines from left-under to top-right.

SlashFillIdem as previous, thick lines.

BkSlashFillFills with thick lines from left-Top to bottom-right.

LtBkSlashFillIdem as previous, normal lines.

HatchFillFills with a hatch-like pattern.

XHatchFillFills with a hatch pattern, rotated 45 degrees.

InterLeaveFill

WideDotFillFills with dots, wide spacing.

CloseDotFillFills with dots, narrow spacing.

UserFillFills with a user-defined pattern.

Errors: None.

See also: SetFillPattern (612), SetWriteMode (616)

12.13.59 SetGraphMode

Synopsis: Set graphical mode

Declaration: `procedure SetGraphMode(Mode: SmallInt)`

Visibility: default

Description: `SetGraphMode` sets the graphical mode and clears the screen.

Errors: None.

See also: InitGraph (607)

12.13.60 SetLineStyle

Synopsis: Set line drawing style

Declaration: `procedure SetLineStyle(LineStyle: Word;Pattern: Word;Thickness: Word)`

Visibility: default

Description: `SetLineStyle` sets the drawing style for lines. You can specify a `LineStyle` which is one of the following pre-defined constants:

SolidLndraws a solid line.

DottedLnDraws a dotted line.

CenterLndraws a non-broken centered line.

DashedLndraws a dashed line.

UserBitLnDraws 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:

NormWidthNormal line width

ThickWidthDouble line width

Errors: None.

See also: GetLineSettings (603), SetWriteMode (616)

12.13.61 SetPalette

Synopsis: Set palette entry using color constant

Declaration: `procedure SetPalette(ColorNum: Word;Color: ShortInt)`

Visibility: default

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

Errors: None.

See also: SetAllPalette (597), SetRGBPalette (597)

12.13.62 SetTextJustify

Synopsis: Set text placement style

Declaration: `procedure SetTextJustify(horiz: Word;vert: Word)`

Visibility: default

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:

LeftTextText is set left of the pointer.

CenterTextText is set centered horizontally on the pointer.

RightTextText 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 :

BottomTextText is placed under the pointer.

CenterTextText is placed centered vertically on the pointer.

TopTextText is placed above the pointer.

Errors: None.

See also: OutText (609), OutTextXY (596)

12.13.63 SetTextStyle

Synopsis: Set text style

Declaration: `procedure SetTextStyle(font: Word;direction: Word;charsize: Word)`

Visibility: default

Description: `SetTextStyle` controls the style of text to be put on the screen. pre-defined constants for `Font` are:

DefaultFontThe default font

TriplexFontA special font

SmallFontA smaller font

SansSerifFontA sans-serif font (like Arial)

GothicFontA gothic font

ScriptFontA script font

SimpleFontA simple font

TSCRFontTerminal screen font

LCOMFont?

EuroFont?

BoldFontA bold typeface font

Pre-defined constants for `Direction` are :

HorizDirWrite horizontal

VertDirWrite vertical

Errors: None.

See also: GetTextSettings (605)

12.13.64 SetUserCharSize

Synopsis: Set user character size for vector font

Declaration: `procedure SetUserCharSize(Multx: Word;Divx: Word;Multy: Word;Divy: Word)`

Visibility: default

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 (615)

12.13.65 SetViewPort

Synopsis: Set the graphical drawing window

Declaration:
```
procedure SetViewPort(X1: SmallInt;Y1: SmallInt;X2: SmallInt;
                      Y2: SmallInt;Clip: Boolean)
```

Visibility: default

Description: Sets the current graphical viewport (window) to the rectangle defined by the top-left corner `(X1,Y1)` and the bottom-right corner `(X2,Y2)`. If `Clip` is true, anything drawn outside the viewport (window) will be clipped (i.e. not drawn). Coordinates specified after this call are relative to the top-left corner of the viewport.

Errors: None.

See also: GetViewSettings (605)

12.13.66 SetWriteMode

Synopsis: Specify binary operation to perform when drawing on screen

Declaration:
```
procedure SetWriteMode(WriteMode: SmallInt)
```

Visibility: default

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:

CopyPutDraw as specified using current bitmask and color

XORPutDraw XOR-ing current bitmask and color

Errors: None.

See also: SetColor (612), SetBkColor (612), SetLineStyle (613), SetFillStyle (613)

12.13.67 TextHeight

Synopsis: Return height (in pixels) of the given string

Declaration:
```
function TextHeight(const TextString: String) : Word
```

Visibility: default

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

Errors: None.

See also: TextWidth (616)

12.13.68 TextWidth

Synopsis: Return width (in pixels) of the given string

Declaration:
```
function TextWidth(const TextString: String) : Word
```

Visibility: default

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

Errors: None.

See also: TextHeight (616)

Chapter 13

Reference for unit 'heaptrc'

13.1 Controlling HeapTrc with environment variables

The HeapTrc unit can be controlled with the `HEAPTRC` environment variable. The contents of this variable controls the initial setting of some constants in the unit. `HEAPTRC` consists of one or more of the following strings, separated by spaces:

keepreleased If this string occurs, then the KeepReleased (620) variable is set to `True`

disabled If this string occurs, then the UseHeapTrace (620) variable is set to `False` and the heap trace is disabled. It does not make sense to combine this value with other values.

nohalt If this string occurs, then the HaltOnError (619) variable is set to `False`, so the program continues executing even in case of a heap error.

log=filename If this string occurs, then the output of heaptrc is sent to the specified `Filename`. (see also SetHeapTraceOutput (622))

The following are valid values for the HEAPTRC variable:

```
HEAPTRC=disabled
HEAPTRC="keepreleased log=heap.log"
HEAPTRC="log=myheap.log nohalt"
```

Note that these strings are case sensitive, and the name of the variable too.

13.2 HeapTrc Usage

All that you need to do is to include heaptrc in the uses clause of your program. Make sure that it is the first unit in the clause, otherwise memory allocated in initialization code of units that precede the heaptrc unit will not be accounted for, causing an incorrect memory usage report.

If you use the `-gh` switch, the compiler will insert the unit by itself, so you don't have to include it in your uses clause.

The below example shows how to use the heaptrc unit.

This is the memory dump shown when running this program in a standard way:

```
Marked memory at 0040FA50 invalid
Wrong size : 128 allocated 64 freed
  0x00408708
  0x0040CB49
  0x0040C481
Call trace for block 0x0040FA50 size 128
  0x0040CB3D
  0x0040C481
```

If you use the lineinfo unit (or use the `-gl` switch) as well, then heaptrc will also give you the filenames and line-numbers of the procedures in the backtrace:

```
Marked memory at 00410DA0 invalid
Wrong size : 128 allocated 64 freed
  0x004094B8
  0x0040D8F9  main,  line 25 of heapex.pp
  0x0040D231
Call trace for block 0x00410DA0 size 128
  0x0040D8ED  main,  line 23 of heapex.pp
  0x0040D231
```

If lines without filename/line-number occur, this means there is a unit which has no debug info included.

13.3 Overview

This document describes the HEAPTRC unit for Free Pascal. It was written by Pierre Muller. It is system independent, and works on all supported systems.

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.

13.4 Constants, types and variables

13.4.1 Constants

```
add_tail : Boolean = true
```

If `add\_tail` is `True` (the default) then a check is also performed on the memory location just behind the allocated memory.

```
HaltOnError : Boolean = true
```

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`.

```
HaltOnNotReleased : Boolean = false
```

HaltOnNotReleased can be set to True to make the DumpHeap (621) procedure halt (exit code 203) the program if any memory was not released when the dump is made. If it is False (the default) then DumpHeap just returns.

```
keepreleased : Boolean = false
```

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.

```
quicktrace : Boolean = true
```

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 True.

```
tracesize = 8
```

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.

```
usecrc : Boolean = true
```

If usecrc is True (the default) then a crc check is performed on locations before and after the allocated memory. This is useful to detect memory overwrites.

```
useheaptrace : Boolean = true
```

This variable must be set at program startup, through the help of an environment variable.

13.4.2 Types

```
tdisplayextrainfoProc = procedure(var ptext: text;p: pointer)
```

The TDisplayExtraInfoType is a procedural type used in the SetHeapExtraInfo (621) call to display a memory location which was previously filled with TFillExtraInfoProc (620)

```
tFillExtraInfoProc = procedure(p: pointer)
```

The TFillExtraInfoProc is a procedural type used in the SetHeapExtraInfo (621) call to fill a memory location with extra data for displaying.

13.5 Procedures and functions

13.5.1 DumpHeap

Synopsis: Dump memory usage report to stderr.

Declaration: `procedure DumpHeap`

Visibility: default

Description: `DumpHeap` dumps to standard output a summary of memory usage. It is called automatically by the heaptrc unit when your program exits (by installing an exit procedure), but it can be called at any time.

Errors: None.

See also: MarkHeap (618)

13.5.2 SetHeapExtraInfo

Synopsis: Store extra information in blocks.

Declaration:
```
procedure SetHeapExtraInfo(size: ptrint;fillproc: tFillExtraInfoProc;
                           displayproc: tdisplayextrainfoProc)
```

Visibility: default

Description: You can use `SetHeapExtraInfo` 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`, `FillProc` will be called, and passed a pointer to the memory reserved.

When dumping the memory summary, the extra info is shown by calling `displayproc` and passing it the memory location which was filled by `fillproc`. It should write the information in readable form to the text file provided in the call to `displayproc`

Errors: You can only call `SetHeapExtraInfo` if no memroy has been allocated yet. If memory was already allocated prior to the call to `SetHeapExtraInfo`, then an error will be displayed on standard error output, and a DumpHeap (621) is executed.

See also: DumpHeap (621), SetHeapTraceOutput (622)

Listing: ./heapex/setinfo.pp

```
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.
```

13.5.3 SetHeapTraceOutput

Synopsis: Specify filename for heap trace output.

Declaration: `procedure SetHeapTraceOutput(const name: String)`

Visibility: default

Description: `SetHeapTraceOutput` sets the filename into which heap trace info will be written. By default information is written to standard output, this function allows you to redirect the information to a file with full filename `name`.

Errors: If the file cannot be written to, errors will occur when writing the trace.

See also: SetHeapExtraInfo (621)

Chapter 14

Reference for unit 'ipc'

14.1 Used units

Table 14.1: Used units by unit 'ipc'

Name	Page
BaseUnix	624
UnixType	624

14.2 Overview

This document describes the IPC unit for Free Pascal. It was written for linux by Michael Van Canneyt. It gives all the functionality of system V Inter-Process Communication: shared memory, semaphores and messages. It works only on the linux operating system.

Many constants here are provided for completeness only, and should under normal circumstances not be used by the programmer.

14.3 Constants, types and variables

14.3.1 Constants

```
IPC_CREAT = 1 shl 9
```

Create if key is nonexistent

```
IPC_EXCL = 2 shl 9
```

fail if key exists

```
IPC_INFO = 3
```

For ipcs call

```
IPC_NOWAIT = 4 shl 9
```

return error on wait

```
IPC_RMID = 0
```

Remove resource

```
IPC_SET = 1
```

set ipc_perm options

```
IPC_STAT = 2
```

get ipc_perm options

```
MSGMAX = 4056
```

Internal Message control code. Do not use

```
MSGMNB = 16384
```

Internal Message control code. Do not use

```
MSGMNI = 128
```

Internal Message control code. Do not use

```
MSG_EXCEPT = 2 shl 12
```

Internal Message control code. Do not use

```
MSG_NOERROR = 1 shl 12
```

Internal Message control code. Do not use

```
SEM_GETALL = 13
```

Semaphore operation: Get all semaphore values

```
SEM_GETNCNT = 14
```

Semaphore operation: Get number of processes waiting for resource.

```
SEM_GETPID = 11
```

Semaphore operation: Get process ID of last operation.

```
SEM_GETVAL = 12
```

Semaphore operation: Get current value of semaphore

```
SEM_GETZCNT = 15
```

Semaphore operation: Get number of processes waiting for semaphores to reach zero

```
SEM_SEMMNI = 128
```

Semaphore operation: ?

```
SEM_SEMMNS = (SEM_SEMMNI * SEM_SEMMSL )
```

Semaphore operation: ?

```
SEM_SEMMSL = 32
```

Semaphore operation: ?

```
SEM_SEMOPM = 32
```

Semaphore operation: ?

```
SEM_SEMVMX = 32767
```

Semaphore operation: ?

```
SEM_SETALL = 17
```

Semaphore operation: Set all semaphore values

```
SEM_SETVAL = 16
```

Semaphore operation: Set semaphore value

```
SEM_UNDO = $1000
```

Constant for use in semop (640)

```
SHM_LOCK = 11
```

This constant is used in the shmctl (642) call.

```
SHM_R = 4 shl 6
```

This constant is used in the shmctl (642) call.

```
SHM_RDONLY = 1 shl 12
```

This constant is used in the shmctl (642) call.

```
SHM_REMAP = 4 shl 12
```

This constant is used in the shmctl (642) call.

```
SHM_RND = 2 shl 12
```

This constant is used in the shmctl (642) call.

```
SHM_UNLOCK = 12
```

This constant is used in the shmctl (642) call.

```
SHM_W = 2 shl 6
```

This constant is used in the shmctl (642) call.

14.3.2 Types

```
key_t = TKey
```

Alias for TKey (628) type

```
msglen_t = culong
```

Message length type

```
msgqnum_t = culong
```

Message queue number type

```
PIPC_Perm = ^TIPC_Perm
```

Pointer to TIPC_Perm (628) record.

```
PMSG = ^TMSG
```

Pointer to TMSG (628) record

```
PMSGbuf = ^TMSGbuf
```

Pointer to TMsgBuf (629) rcord

```
PMSGinfo = ^TMSGinfo
```

Pointer to TMSGinfo (629) record

```
PMSQid_ds = ^TMSQid_ds
```

Pointer to TMSQid_ds (629)

```
PSEMbuf = ^TSEMbuf
```

Pointer to TSembuf (629) record.

```
PSEMid_ds = ^TSEMid_ds
```

Pointer to TSEMid_ds (630) record.

```
PSEMinfo = ^TSEMinfo
```

Pointer to TSEMinfo (630) record.

```
PSEMun = ^TSEMun
```

Pointer to TSEMun (630) record

```
PShmid_DS = ^TShmid_ds
```

Pointer to TSHMid_ds (630) record.

```
PSHMinfo = ^TSHMinfo
```

```
TIPC_Perm = record
  key : TKey;
  uid : uid_t;
  gid : gid_t;
  cuid : uid_t;
  cgid : gid_t;
  mode : mode_t;
  seq : cushort;
end
```

`TIPC_Perm` is used in all IPC systems to specify the permissions. It should never be used directly.

```
TKey = cint
```

Type returned by the ftok (631) key generating function.

```
TMSG = record
  msg_next : PMSG;
  msg_type : LongInt;
  msg_spot : PChar;
  msg_stime : LongInt;
  msg_ts : Integer;
end
```

Record used in the handling of message queues. Do not use directly.

```
TMSGbuf = record
  mtype : clong;
  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.

```
TMSGinfo = record
  msgpool : cint;
  msgmap : cint;
  msgmax : cint;
  msgmnb : cint;
  msgmni : cint;
  msgssz : cint;
  msgtql : cint;
  msgseg : cushort;
end
```

Internal message system record. Do not use directly.

```
TMSQid_ds = record
  msg_perm : TIPC_Perm;
  msg_first : PMSG;
  msg_last : PMSG;
  msg_stime : time_t;
  msg_rtime : time_t;
  msg_ctime : time_t;
  msg_cbytes : Word;
  msg_qnum : Word;
  msg_qbytes : Word;
  msg_lspid : ipc_pid_t;
  msg_lrpid : ipc_pid_t;
end
```

This record should never be used directly, it is an internal kernel record. It's fields may change at any time.

```
TSEMbuf = record
  sem_num : cushort;
  sem_op : cshort;
  sem_flg : cshort;
end
```

The TSEMbuf record is used in the semop (640) call, and is used to specify which operations you want to do.

```
TSEMid_ds = record
  sem_perm : TIPC_Perm;
  sem_otime : time_t;
  sem_ctime : time_t;
  sem_base : pointer;
  sem_pending : pointer;
```

```
  sem_pending_last : pointer;
  undo : pointer;
  sem_nsems : cushort;
end
```

Structure returned by the semctl (635) call, contains all data of a semahore

```
TSEMinfo = record
  semmap : cint;
  semmni : cint;
  semmns : cint;
  semmnu : cint;
  semmsl : cint;
  semopm : cint;
  semume : cint;
  semusz : cint;
  semvmx : cint;
  semaem : cint;
end
```

Internal semaphore system record. Do not use.

```
TSEMun = record
end
```

Record used in semctl (635) call.

```
TShmid_ds = record
  shm_perm : TIPC_Perm;
  shm_segsz : cint;
  shm_atime : time_t;
  shm_dtime : time_t;
  shm_ctime : time_t;
  shm_cpid : ipc_pid_t;
  shm_lpid : ipc_pid_t;
  shm_nattch : Word;
  shm_npages : Word;
  shm_pages : Pointer;
  attaches : pointer;
end
```

Record used in the shmctl (642) call to set or retrieve settings for shared memory.

```
TSHMinfo = record
  shmmax : cint;
  shmmin : cint;
  shmmni : cint;
  shmseg : cint;
  shmall : cint;
end
```

Record used by the shared memory system, Do not use directly.

14.4 Procedures and functions

14.4.1 ftok

Synopsis: Create token from filename

Declaration: `function ftok(Path: pchar;ID: cint) : TKey`

Visibility: default

Description: `ftok` returns a key that can be used in a semget (640)shmget (644) or msgget (634) 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`

For an example, see msgctl (631), semctl (635) or shmctl (642).

Errors: `ftok` returns -1 if the file in `Path` doesn't exist.

See also: semget (640), shmget (644), msgget (634)

14.4.2 msgctl

Synopsis: Perform various operations on a message queue

Declaration: `function msgctl(msqid: cint;cmd: cint;buf: PMSQid_ds) : cint`

Visibility: default

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 (634), msgsnd (635), msgrcv (634)

Listing: ./ipcex/msgtool.pp

```
program msgtool;

Uses ipc, baseunix;

Type
```

```
  PMyMsgBuf = ^TMyMsgBuf;
  TMyMsgBuf = record
    mtype : Longint;
    mtext : string[255];
  end;

Procedure DoError (Const Msg : string);

begin
  Writeln (msg,' returned an error : ',fpgeterrno);
  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  msgsnd(Id,PMsgBuf(@Buf),256,0)=-1 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)<>-1 then
    Writeln ('Type : ',buf.mtype,' Text : ',buf.mtext)
  else
    DoError ('msgrcv');
end;

Procedure RemoveQueue ( ID : Longint);

begin
  If msgctl (id,IPC_RMID,Nil)<>-1 then
    Writeln ('Removed Queue with id ',Id);
end;

Procedure ChangeQueueMode (ID,mode : longint);

Var QueueDS : TMSQid_ds;

begin
  If  msgctl (Id,IPC_STAT,@QueueDS)=-1 then
    DoError ('msgctl : stat');
  Writeln ('Old permissions : ',QueueDS.msg_perm.mode);
  QueueDS.msg_perm.mode:=Mode;
  if msgctl (ID,IPC_SET,@QueueDS)=0 then
    Writeln ('New permissions : ',QueueDS.msg_perm.mode)
  else
```

```
    DoError ('msgctl : IPC_SET');
end;

procedure usage;

begin
  WriteIn ('Usage : msgtool s(end)    <type> <text> (max 255 characters)');
  WriteIn ('                r(eceive) <type>');
  WriteIn ('                d(elete)');
  WriteIn ('                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;

const ipckey = '.'#0;

begin
  If Paramcount<1 then Usage;
  key := Ftok(@ipckey[1],ord('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.
```

14.4.3 msgget

Synopsis: Return message queue ID, possibly creating the queue

Declaration:
```
function msgget(key: TKey;msgflg: cint) : cint
```

Visibility: default

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.

For an example, see msgctl (631).

Errors: On error, -1 is returned, and `IPCError` is set.

See also: ftok (631), msgsnd (635), msgrcv (634), msgctl (631)

14.4.4 msgrcv

Synopsis: Retrieve a message from the queue

Declaration:
```
function msgrcv(msqid: cint;msgp: PMSGbuf;msgsz: size_t;msgtyp: cint;
                msgflg: cint) : cint
```

Visibility: default

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:

0 No special meaning.

IPC_NOWAIT if 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.

For an example, see msgctl (631).

Errors: In case of error, `False` is returned, and `IPCerror` is set.

See also: msgget (634), msgsnd (635), msgctl (631)

14.4.5 msgsnd

Synopsis: Send a message to the messague queue

Declaration:
```
function msgsnd(msqid: cint;msgp: PMSGbuf;msgsz: size_t;msgflg: cint)
                      : cint
```

Visibility: default

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):

0 No 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.

For an example, see msgctl (631).

Errors: In case of error, the call returns `False`, and `IPCerror` is set.

See also: msgget (634), msgrcv (634), msgctl (631)

14.4.6 semctl

Synopsis: Perform various control operations on a semaphore set

Declaration:
```
function semctl(semid: cint;semnum: cint;cmd: cint;var arg: TSEMun)
                      : cint
```

Visibility: default

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.

GETALL In 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 \var{SizeOf(Word)* Number of semaphores in the set}. This call will then fill the memory array with all the values of the semaphores.

GETNCNT This will fill the `val` field of the `arg` union with the number of processes waiting for resources.

GETPID `semctl` returns the process ID of the process that performed the last semop (640) 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.

SETALL In 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 \var{SizeOf(Word)* Number of semaphores in the set}. This call will then set the values of the semaphores from the memory array.

SETVAL This 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 (640), semop (640)

Listing: ./ipcex/semtool.pp

```
Program semtool;

{ Program to demonstrat the use of semaphores }

Uses ipc,baseunix;

Const MaxSemValue = 5;

Procedure DoError (Const Msg : String);

begin
  WriteIn ('Error : ',msg,' Code : ',fpgeterrno);
  Halt(1);
end;

Function getsemval (ID,Member : longint) : longint;

Var S : TSEMun;

begin
  GetSemVal:=SemCtl(id,member,SEM_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
// the semmsl constant seems kernel specific
{  If members>semmsl then
    DoError ('Sorry, maximum number of semaphores in set exceeded');
}
  WriteIn ('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,SEM_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 semop(Id,@lock,1)=-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 semop(Id,@unlock,1)=-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;


const ipckey='.'#0;

begin
  If ParamCount<1 then USage;
  key:=ftok(@ipckey[1],ORD('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.
```

14.4.7 semget

Synopsis: Return the ID of a semaphore set, possibly creating the set

Declaration: `function semget(key: TKey;nsems: cint;semflg: cint) : cint`

Visibility: default

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 (631), semop (640), semctl (635)

14.4.8 semop

Synopsis: Perform semaphore operation.

Declaration: `function semop(semid: cint;sops: PSEMbuf;nsops: cuint) : cint`

Visibility: default

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 (629) 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`: A positive number is simply added to the current value of the semaphore. If 0 (zero) is specified, then the process is suspended until the specified semaphore reaches zero. 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 (640), semctl (635)

14.4.9 shmat

Synopsis: Attach a shared memory block.

Declaration: `function shmat(shmid: cint;shmaddr: pointer;shmflg: cint) : pointer`

Visibility: default

Description: `shmat` attaches a shared memory block with identified `shmid` to the current process. The function returns a pointer to the shared memory block.

If `shmaddr` is `Nil`, then the system chooses a free unmapped memory region, as high up in memory space as possible.

If `shmaddr` is non-nil, and `SHM_RND` is in `shmflg`, then the returned address is `shmaddr`, rounded down to `SHMLBA`. If `SHM_RND` is not specified, then `shmaddr` must be a page-aligned address.

The parameter `shmflg` can be used to control the behaviour of the `shmat` call. It consists of a ORed combination of the following constants:

SHM_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.

For an example, see shmctl (642).

Errors: If an error occurs, -1 is returned, and `IPCerror` is set.

See also: shmget (644), shmdt (644), shmctl (642)

14.4.10 shmctl

Synopsis: Perform control operations on a shared memory block.

Declaration: `function shmctl(shmid: cint;cmd: cint;buf: PShmid_DS) : cint`

Visibility: default

Description: `shmctl` performs various operations on the shared memory block identified by identifier `shmid`.

The `buf` parameter points to a `TSHMid_ds` record. The `cmd` parameter is used to pass which operation is to be performed. It can have one of the following values :

IPC_STAT `shmctl` fills the `TSHMid_ds` record that `buf` points to with the available information about the shared memory block.

IPC_SET applies the values in the `ipc_perm` record that `buf` points to, to the shared memory block.

IPC_RMID the shared memory block is destroyed (after all processes to which the block is attached, have detached from it).

If successful, the function returns `True`, `False` otherwise.

Errors: If an error occurs, the function returns `False`, and `IPCerror` is set.

See also: shmget (644), shmat (641), shmdt (644)

Listing: ./ipcex/shmtool.pp

```
Program shmtool;

uses ipc,strings,Baseunix;

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 shmctl (shmid,IPC_STAT,@data)=-1 then
    begin
     writeln ('Error : shmctl :',fpgeterrno);
     halt(1);
    end;
  writeln ('Old permissions : ',data.shm_perm.mode);
  data.shm_perm.mode:=m;
  If shmctl (shmid,IPC_SET,@data)=-1 then
    begin
    writeln ('Error : shmctl :',fpgeterrno);
    halt(1);
    end;
  writeln ('New permissions : ',data.shm_perm.mode);
end;

const ftokpath = '.'#0;

begin
  if paramcount<1 then usage;
   key := ftok (pchar(@ftokpath[1]),ord('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 !',fpgeterrno);
      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 !',fpgeterrno);
    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.
```

14.4.11 shmdt

Synopsis: Detach shared memory block.

Declaration: `function shmdt(shmaddr: pointer) : cint`

Visibility: default

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 (641).

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 (644), shmat (641), shmctl (642)

14.4.12 shmget

Synopsis: Return the ID of a shared memory block, possibly creating it

Declaration: `function shmget(key: TKey;size: size_t;flag: cint) : cint`

Visibility: default

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` bytes in it.

Errors: On error, -1 is returned, and `IPCError` is set.

Chapter 15

Reference for unit 'keyboard'

15.1 Unix specific notes

On Unix, applications run on a "terminal", and the application writes to the screen and reads from the keyboard by communicating with the terminal. Unix keyboard handling is mostly backward compatible with the DEC vt100 and vt220 terminals from tens of years ago. The vt100 and vt220 had very different keyboards than todays PC's and this is where the problems start. To make it worse the protocol of both terminals has not been very well designed.

Because of this, the keyboard unit on Unix operating systems does a best effort to provide keyboard functionality. An implementation with full keyboard facilities like on other operating systems is not possible.

The exception is the Linux kernel. The terminal emulation of the Linux kernel is from a PC keyboard viewpoint hopeless as well, but unlike other terminal emulators it is configurable. On the Linux console, the Free Pascal keyboard unit tries to implement full functionality.

Users of applications using the keyboard unit should expect the following:

- Full functionality on the Linux console. It must be the bare console, SSH into another machine will kill the full functionality.
- Limited functionality otherwise.

Notes about Linux full functionality:

- The keyboard is reprogrammed. If the keyboard is for whatever reason not restored in its original state, please load your keymap to reinitialize it.
- Alt+function keys generate keycodes for those keys. To switch virtual consoles, use ctrl+alt+function key.
- Unlike what you're used to with other Unix software, escape works as you intuitively expect, it generates the keycode for an escape key **without a delay.**

The limited functionality does include these quirks:

- Escape must be pressed two times before it has effect.
- On the Linux console, when the users runs the program by logging into another machine:
  - Shift+F1 and Shift+F12 will generate keycodes for F11 and F12.

  - Shift+arrow keys, shift+ins, shift+del, shift+home, shift+end do not work. The same is true about the control and alt combinations.
  - Alt+function keys will switch virtual consoles instead of generating the right key sequences.
  - Ctrl+function keys will generate the keycodes for the function keys without ctrl

- In Xterm:
  - Shift+insert pastes the x clipboard, no keycode will be generated.

- In Konsole:
  - Shift+insert pastes the x clipboard, no keycode will be generated.
  - Shift+arrow keys doesn't work, nor does ctrl+arrow keys

If you have a non-standard terminal, some keys may not work at all. When in limited functionality mode, the user can work around using an escape prefix:

- Esc+1 = F1, Esc+2 = F2.
- Esc before another key is equal to alt+key.

In such cases, if the terminal does output an escape sequence for those keys, please submit a bug report so we can add them.

15.2 Writing a keyboard driver

Writing a keyboard driver means that hooks must be created for most of the keyboard unit functions. The `TKeyBoardDriver` record contains a field for each of the possible hooks:

```
TKeyboardDriver = Record
  InitDriver : Procedure;
  DoneDriver : Procedure;
  GetKeyEvent : Function : TKeyEvent;
  PollKeyEvent : Function : TKeyEvent;
  GetShiftState : Function : Byte;
  TranslateKeyEvent : Function (KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUniCode: Function (KeyEvent: TKeyEvent): TKeyEvent;
end;
```

The meaning of these hooks is explained below:

InitDriver Called to initialize and enable the driver. Guaranteed to be called only once. This should initialize all needed things for the driver.

DoneDriver Called to disable and clean up the driver. Guaranteed to be called after a call to `initDriver`. This should clean up all things initialized by `InitDriver`.

GetKeyEvent Called by GetKeyEvent (654). Must wait for and return the next key event. It should NOT store keys.

PollKeyEvent Called by PollKeyEvent (659). It must return the next key event if there is one. Should not store keys.

GetShiftState Called by PollShiftStateEvent (660). Must return the current shift state.

TranslateKeyEvent Should translate a raw key event to a cOrrect key event, i.e. should fill in the shiftstate and convert function key scancodes to function key keycodes. If the `TranslateKeyEvent` is not filled in, a default translation function will be called which converts the known scancodes from the tables in the previous section to a correct keyevent.

TranslateKeyEventUniCode Should translate a key event to a unicode key representation.

Strictly speaking, only the `GetKeyEvent` and `PollKeyEvent` hooks must be implemented for the driver to function correctly.

The example unit demonstrates how a keyboard driver can be installed. It takes the installed driver, and hooks into the `GetKeyEvent` function to register and log the key events in a file. This driver can work on top of any other driver, as long as it is inserted in the `uses` clause *after* the real driver unit, and the real driver unit should set the driver record in its initialization section.

Note that with a simple extension of this unit could be used to make a driver that is capable of recording and storing a set of keyboard strokes, and replaying them at a later time, so a 'keyboard macro' capable driver. This driver could sit on top of any other driver.

15.3 Keyboard scan codes

Special physical keys are encoded with the DOS scan codes for these keys in the second byte of the TKeyEvent (652) type. A complete list of scan codes can be found in the below table. This is the list of keys that is used by the default key event translation mechanism. When writing a keyboard driver, either these constants should be returned by the various key event functions, or the `TranslateKeyEvent` hook should be implemented by the driver.

A list of scan codes for special keys and combinations with the SHIFT, ALT and CTRL keys can be found in the following table: They are for quick reference only.

15.4 Overview

The Keyboard unit implements a keyboard access layer which is system independent. It can be used to poll the keyboard state and wait for certain events. Waiting for a keyboard event can be done with the GetKeyEvent (654) function, which will return a driver-dependent key event. This key event can be translated to a interpretable event by the TranslateKeyEvent (663) function. The result of this function can be used in the other event examining functions.

A custom keyboard driver can be installed using the SetKeyboardDriver (662) function. The current keyboard driver can be retrieved using the GetKeyboardDriver (654) function. The last section of this chapter demonstrates how to make a keyboard driver.

15.5 Constants, types and variables

15.5.1 Constants

```
AltPrefix : Byte = 0
```

Alt key name index.

```
CtrlPrefix : Byte = 0
```

Alt key name index.

```
errKbdBase = 1010
```

Base of keyboard routine error reporting constants.

```
errKbdInitError = errKbdBase + 0
```

Failed to initialize keyboard driver

```
errKbdNotImplemented = errKbdBase + 1
```

Keyboard driver not implemented.

```
kbAlt = 8
```

Alt key modifier

```
kbASCII = $00
```

Ascii code key event

```
kbCtrl = 4
```

Control key modifier

```
kbdApps = $FF17
```

Application key (popup-menu) pressed.

```
kbdDelete = $FF2A
```

Delete key pressed

```
kbdDown = $FF27
```

Arrow down key pressed

```
kbdEnd = $FF26
```

End key pressed

```
kbdF1 = $FF01
```

F1 function key pressed.

```
kbdF10 = $FF0A
```

F10 function key pressed.

```
kbdF11 = $FF0B
```

F12 function key pressed.

```
kbdF12 = $FF0C
```

F12 function key pressed.

```
kbdF13 = $FF0D
```

F13 function key pressed.

```
kbdF14 = $FF0E
```

F14 function key pressed.

```
kbdF15 = $FF0F
```

F15 function key pressed.

```
kbdF16 = $FF10
```

F16 function key pressed.

```
kbdF17 = $FF11
```

F17 function key pressed.

```
kbdF18 = $FF12
```

F18 function key pressed.

```
kbdF19 = $FF13
```

F19 function key pressed.

```
kbdF2 = $FF02
```

F2 function key pressed.

```
kbdF20 = $FF14
```

F20 function key pressed.

```
kbdF3 = $FF03
```

F3 function key pressed.

```
kbdF4 = $FF04
```

F4 function key pressed.

```
kbdF5 = $FF05
```

F5 function key pressed.

```
kbdF6 = $FF06
```

F6 function key pressed.

```
kbdF7 = $FF07
```

F7 function key pressed.

```
kbdF8 = $FF08
```

F8 function key pressed.

```
kbdF9 = $FF09
```

F9 function key pressed.

```
kbdHome = $FF20
```

Home key pressed

```
kbdInsert = $FF29
```

Insert key pressed

```
kbdLeft = $FF23
```

Arrow left key pressed

```
kbdLWin = $FF15
```

Left windows key pressed.

```
kbdMiddle = $FF24
```

Middle key pad key pressed (numerical 5)

```
kbdPgDn = $FF28
```

Page down key pressed

```
kbdPgUp = $FF22
```

Page Up key pressed

```
kbdRight = $FF25
```

Arrow right key pressed

```
kbdRWin = $FF16
```

Right windows key pressed.

```
kbdUp = $FF21
```

Arrow up key pressed

```
kbFnKey = $02
```

function key pressed.

```
kbLeftShift = 1
```

Left shift key modifier

```
kbPhys = $03
```

Physical key code event

```
kbReleased = $04
```

Key release event

```
kbRightShift = 2
```

Right shift key modifier

```
kbShift = kbLeftShift or kbRightShift
```

Shift key modifier

```
kbUniCode = $01
```

Unicode code key event

```
SAnd : String = 'AND'
```

This constant is used as the 'And' word in key descriptions. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
ShiftPrefix : Byte = 0
```

Shift key name index.

```
SKeyPad : Array[0..($FF2F-kbdHome)] of String = ('Home','Up','PgUp','Left','Middle',
```

This constant describes all keypad keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SLeftRight : Array[1..2] of String = ('LEFT','RIGHT' )
```

This constant contains strings to describe left and right keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SScanCode : String = 'Key with scancode '
```

This constant contains a string to denote a scancode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SShift : Array[1..3] of String = ('SHIFT','CTRL','ALT' )
```

This constant describes the various modifier keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SUnicodeChar : String = 'Unicode character '
```

This constant contains a string to denote a unicode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SUnknownFunctionKey : String = 'Unknown function key : '
```

This constant contains a string to denote that an unknown function key was found. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

15.5.2 Types

```
TKeyboardDriver = record
  InitDriver : procedure;
  DoneDriver : procedure;
  GetKeyEvent : function : TKeyEvent;
  PollKeyEvent : function : TKeyEvent;
  GetShiftState : function : Byte;
  TranslateKeyEvent : function(KeyEvent: TKeyEvent) : TKeyEvent;
  TranslateKeyEventUniCode : function(KeyEvent: TKeyEvent) : TKeyEvent;
end
```

The `TKeyboardDriver` record can be used to install a custom keyboard driver with the SetKeyboardDriver (662) function.

The various fields correspond to the different functions of the keyboard unit interface. For more information about this record see kbddriver (646)

```
TKeyEvent = Cardinal
```

The `TKeyEvent` type is the base type for all keyboard events.

The key stroke is encoded in the 4 bytes of the `TKeyEvent` type. The various fields of the key stroke encoding can be obtained by typecasting the `TKeyEvent` type to the TKeyRecord (653) type.

```
TKeyRecord = packed record
  KeyCode : Word;
  ShiftState : Byte;
  Flags : Byte;
end
```

The structure of a `TKeyRecord` structure is explained in the following table:

The shift-state can be checked using the various shift-state constants, and the flags in the last byte can be checked using one of the kbASCII, kbUniCode, kbFnKey, kbPhys, kbReleased constants.

If there are two keys returning the same char-code, there's no way to find out which one was pressed (Gray+ and Simple+). If it needs to be known which was pressed, the untranslated keycodes must be used, but these are system dependent. System dependent constants may be defined to cover those, with possibily having the same name (but different value).

15.6 Procedures and functions

15.6.1 AddSequence

Declaration: `procedure AddSequence(const St: String;AChar: Byte;AScan: Byte)`

Visibility: default

15.6.2 DoneKeyboard

Synopsis: Deactivate keyboard driver.

Declaration: `procedure DoneKeyboard`

Visibility: default

Description: `DoneKeyboard` de-initializes the keyboard interface if the keyboard driver is active. If the keyboard driver is not active, the function does nothing.

This will cause the keyboard driver to clear up any allocated memory, or restores the console or terminal the program was running in to its initial state before the call to InitKeyBoard (658). This function should be called on program exit. Failing to do so may leave the terminal or console window in an unusable state. Its exact action depends on the platform on which the program is running.

On Unix the default keyboard driver restores the line ending of system.output to #10.

For an example, see most other functions.

Errors: None.

See also: InitKeyBoard (658)

15.6.3 FindSequence

Declaration: `function FindSequence(const St: String;var AChar: Byte;var Ascan: Byte) : Boolean`

Visibility: default

15.6.4 FunctionKeyName

Synopsis: Return string representation of a function key code.

Declaration: `function FunctionKeyName(KeyCode: Word) : String`

Visibility: default

Description: `FunctionKeyName` returns a string representation of the function key with code `KeyCode`. This can be an actual function key, or one of the cursor movement keys.

Errors: In case `KeyCode` does not contain a function code, the `SUnknownFunctionKey` string is returned, appended with the `KeyCode`.

See also: ShiftStateToString (663), KeyEventToString (659)

Listing: ./kbdex/ex8.pp

```
Program Example8;

{ Program to demonstrate the FunctionKeyName function. }

Uses keyboard;

Var
  K : TkeyEvent;

begin
  InitKeyboard;
  Writeln('Press function keys, press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    If IsFunctionKey(k) then
      begin
      Write('Got function key : ');
      Writeln(FunctionKeyName(TkeyRecord(K).KeyCode));
      end;
  Until (GetKeyEventChar(K)='q');
  DoneKeyboard;
end.
```

15.6.5 GetKeyboardDriver

Synopsis: Return the current keyboard driver record.

Declaration: `procedure GetKeyboardDriver(var Driver: TKeyboardDriver)`

Visibility: default

Description: `GetKeyBoardDriver` returns in `Driver` the currently active keyboard driver. This function can be used to enhance an existing keyboarddriver.

For more information on getting and setting the keyboard driver kbddriver (646).

Errors: None.

See also: SetKeyboardDriver (662)

15.6.6 GetKeyEvent

Synopsis: Get the next raw key event, wait if needed.

Declaration: `function GetKeyEvent : TKeyEvent`

Visibility: default

Description: `GetKeyEvent` returns the last keyevent if one was stored in `PendingKeyEvent`, or waits for one if none is available. A non-blocking version is available in PollKeyEvent (659).

The returned key is encoded as a `TKeyEvent` type variable, and is normally the physical key scan code, (the scan code is driver dependent) which can be translated with one of the translation functions TranslateKeyEvent (663) or TranslateKeyEventUniCode (663). See the types section for a description of how the key is described.

Errors: If no key became available (e.g. when the driver does not support it), 0 is returned.

See also: PutKeyEvent (661), PollKeyEvent (659), TranslateKeyEvent (663), TranslateKeyEventUniCode (663)

Listing: ./kbdex/ex1.pp

```
program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    Write('Got key event with ');
    Case GetKeyEventFlags(K) of
      kbASCII     : Writeln('ASCII key');
      kbUniCode   : Writeln('Unicode key');
      kbFnKey     : Writeln('Function key');
      kbPhys      : Writeln('Physical key');
      kbReleased  : Writeln('Released key event');
    end;
    Writeln('Got key : ',KeyEventToString(K));
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

15.6.7 GetKeyEventChar

Synopsis: Get the character key part of a key event.

Declaration: `function GetKeyEventChar(KeyEvent: TKeyEvent) : Char`

Visibility: default

Description: `GetKeyEventChar` returns the charcode part of the given `KeyEvent`, if it contains a translated character key keycode. The charcode is simply the ascii code of the character key that was pressed.

It returns the null character if the key was not a character key, but e.g. a function key.

For an example, see GetKeyEvent (654)

Errors: None.

See also: GetKeyEventUniCode (657), GetKeyEventShiftState (657), GetKeyEventFlags (656), GetKeyEventCode (656), GetKeyEvent (654)

15.6.8 GetKeyEventCode

Synopsis: Translate function key part of a key event code.

Declaration: `function GetKeyEventCode(KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventCode` returns the translated function keycode part of the given KeyEvent, if it contains a translated function key.

If the key pressed was not a function key, the null character is returned.

Errors: None.

See also: GetKeyEventUniCode (657), GetKeyEventShiftState (657), GetKeyEventFlags (656), GetKeyEventChar (655), GetKeyEvent (654)

Listing: ./kbdex/ex2.pp

```
Program Example2;

{ Program to demonstrate the GetKeyEventCode function. }

Uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press function keys, or press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    If (GetKeyEventFlags(K)<>KbfnKey) then
      Writeln('Not a function key')
    else
      begin
      Write('Got key (',GetKeyEventCode(K));
      Writeln(') : ',KeyEventToString(K));
      end;
  Until (GetKeyEventChar(K)='q');
  DoneKeyboard;
end.
```

15.6.9 GetKeyEventFlags

Synopsis: Extract the flags from a key event.

Declaration: `function GetKeyEventFlags(KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventFlags` returns the flags part of the given `KeyEvent`.

For an example, see GetKeyEvent (654)

Errors: None.

See also: GetKeyEventUniCode (657), GetKeyEventShiftState (657), GetKeyEventCode (656), GetKeyEventChar (655), GetKeyEvent (654)

15.6.10 GetKeyEventShiftState

Synopsis: Return the current state of the shift keys.

Declaration: `function GetKeyEventShiftState(KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventShiftState` returns the shift-state values of the given `KeyEvent`. This can be used to detect which of the modifier keys `Shift`, `Alt` or `Ctrl` were pressed. If none were pressed, zero is returned.

Note that this function does not always return expected results; In a unix X-Term, the modifier keys do not always work.

Errors: None.

See also: GetKeyEventUniCode (657), GetKeyEventFlags (656), GetKeyEventCode (656), GetKeyEventChar (655), GetKeyEvent (654)

Listing: ./kbdex/ex3.pp

```
Program Example3;

{ Program to demonstrate the GetKeyEventShiftState function. }

Uses keyboard;

Var
  K : TKeyEvent;
  S : Byte;

begin
  InitKeyBoard;
  Write('Press keys combined with CTRL/SHIFT/ALT');
  Writeln(', or press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    S:=GetKeyEventShiftState(K);
    If (S=0) then
      Writeln('No special keys pressed')
    else
      begin
      Writeln('Detected special keys : ',ShiftStateToString(K,False));
      Writeln('Got key : ',KeyEventToString(K));
      end;
  Until (GetKeyEventChar(K)='q');
  DoneKeyboard;
end.
```

15.6.11 GetKeyEventUniCode

Synopsis: Return the unicode key event.

Declaration: `function GetKeyEventUniCode(KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventUniCode` returns the unicode part of the given `KeyEvent` if it contains a translated unicode character.

Errors: None.

See also: GetKeyEventShiftState (657), GetKeyEventFlags (656), GetKeyEventCode (656), GetKeyEventChar (655), GetKeyEvent (654)

15.6.12 InitKeyboard

Synopsis: Initialize the keyboard driver.

Declaration: `procedure InitKeyboard`

Visibility: default

Description: `InitKeyboard` initializes the keyboard driver. If the driver is already active, it does nothing. When the driver is initialized, it will do everything necessary to ensure the functioning of the keyboard, including allocating memory, initializing the terminal etc.

This function should be called once, before using any of the keyboard functions. When it is called, the DoneKeyboard (653) function should also be called before exiting the program or changing the keyboard driver with SetKeyboardDriver (662).

On Unix, the default keyboard driver sets terminal in raw mode. In raw mode the line feed behaves as an actual linefeed, i.e. the cursor is moved down one line. while the x coordinate does not change. To compensate, the default keyboard sets driver line ending of system.output to #13#10.

For an example, see most other functions.

Errors: None.

See also: DoneKeyboard (653), SetKeyboardDriver (662)

15.6.13 IsFunctionKey

Synopsis: Check whether a given event is a function key event.

Declaration: `function IsFunctionKey(KeyEvent: TKeyEvent) : Boolean`

Visibility: default

Description: `IsFunctionKey` returns `True` if the given key event in `KeyEvent` was a function key or not.

Errors: None.

See also: GetKeyEvent (654)

Listing: ./kbdex/ex7.pp

```
program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
```

```
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  Repeat
    K:=GetKeyEvent;
       K:=TranslateKeyEvent(K);
    If IsFunctionKey(K) then
      Writeln('Got function key : ',KeyEventToString(K))
    else
      Writeln('not a function key.');
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

15.6.14 KeyEventToString

Synopsis: Return a string describing the key event.

Declaration: `function KeyEventToString(KeyEvent: TKeyEvent) : String`

Visibility: default

Description: `KeyEventToString` translates the key event in `KeyEvent` to a human-readable description of the pressed key. It will use the constants described in the constants section to do so.

For an example, see most other functions.

Errors: If an unknown key is passed, the scancode is returned, prefixed with the `SScanCode` string.

See also: FunctionKeyName (653), ShiftStateToString (663)

15.6.15 KeyPressed

Synopsis: Check event queue for key press

Declaration: `function KeyPressed : Boolean`

Visibility: default

Description: `KeyPressed` checks the keyboard event queue to see whether a key event is present, and returns `True` if a key event is available. This function simply calls PollKeyEvent (659) and checks for a valid result.

Errors: None.

See also: PollKeyEvent (659), GetKeyEvent (654)

15.6.16 PollKeyEvent

Synopsis: Get next key event, but does not wait.

Declaration: `function PollKeyEvent : TKeyEvent`

Visibility: default

Description: `PollKeyEvent` checks whether a key event is available, and returns it if one is found. If no event is pending, it returns 0.

Note that this does not remove the key from the pending keys. The key should still be retrieved from the pending key events list with the GetKeyEvent (654) function.

Errors: None.

See also: PutKeyEvent (661), GetKeyEvent (654)

Listing: ./kbdex/ex4.pp

```
program example4;

{ This program demonstrates the PollKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
      K:=GetKeyEvent;
      K:=TranslateKeyEvent(K);
      writeln;
      Writeln('Got key : ',KeyEventToString(K));
      end
    else
      write('.');
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

15.6.17 PollShiftStateEvent

Synopsis: Check current shift state.

Declaration: `function PollShiftStateEvent : TKeyEvent`

Visibility: default

Description: `PollShiftStateEvent` returns the current shiftstate in a keyevent. This will return 0 if there is no key event pending.

Errors: None.

See also: PollKeyEvent (659), GetKeyEvent (654)

Listing: ./kbdex/ex6.pp

```
program example6;

{ This program demonstrates the PollShiftStateEvent function }

uses keyboard;

Var
  K : TKeyEvent;
```

```
begin
  InitKeyBoard;
    Writeln('Press keys, press "q" to end.');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
      K:=PollShiftStateEvent;
      Writeln('Got shift state : ',ShiftStateToString(K,False));
      // Consume the key.
      K:=GetKeyEvent;
      K:=TranslateKeyEvent(K);
      end
{     else
      write('.')};
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

15.6.18 PutKeyEvent

Synopsis: Put a key event in the event queue.

Declaration: `procedure PutKeyEvent(KeyEvent: TKeyEvent)`

Visibility: default

Description: `PutKeyEvent` adds the given `KeyEvent` to the input queue. Please note that depending on the implementation this can hold only one value, i.e. when calling `PutKeyEvent` multiple times, only the last pushed key will be remembered.

Errors: None

See also: PollKeyEvent (659), GetKeyEvent (654)

Listing: ./kbdex/ex5.pp

```
program example5;

{ This program demonstrates the PutKeyEvent function }

uses keyboard;

Var
  K,k2 : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  K2:=0;
  Repeat
    K:=GetKeyEvent;
    If k<>0 then
      begin
      if (k2 mod 2)=0 then
        K2:=K+1
      else
```

```
        K2:=0;
      K:=TranslateKeyEvent(K);
        Writeln('Got key : ',KeyEventToString(K));
      if (K2<>0) then
        begin
        PutKeyEvent(k2);
        K2:=TranslateKeyEVent(K2);
        Writeln('Put key : ',KeyEventToString(K2))
        end
      end
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

15.6.19 RawReadKey

Declaration: `function RawReadKey : Char`

Visibility: default

15.6.20 RawReadString

Declaration: `function RawReadString : String`

Visibility: default

15.6.21 RestoreStartMode

Declaration: `procedure RestoreStartMode`

Visibility: default

15.6.22 SetKeyboardDriver

Synopsis: Set a new keyboard driver.

Declaration: `function SetKeyboardDriver(const Driver: TKeyboardDriver) : Boolean`

Visibility: default

Description: `SetKeyBoardDriver` sets the keyboard driver to `Driver`, if the current keyboard driver is not yet initialized. If the current keyboard driver is initialized, then `SetKeyboardDriver` does nothing. Before setting the driver, the currently active driver should be disabled with a call to DoneKeyboard (653).

The function returns `True` if the driver was set, `False` if not.

For more information on setting the keyboard driver, see kbddriver (646).

Errors: None.

See also: GetKeyboardDriver (654), DoneKeyboard (653)

15.6.23 ShiftStateToString

Synopsis: Return description of key event shift state

Declaration:
```
function ShiftStateToString(KeyEvent: TKeyEvent;UseLeftRight: Boolean)
                                  : String
```

Visibility: default

Description: `ShiftStateToString` returns a string description of the shift state of the key event `KeyEvent`. This can be an empty string.

The shift state is described using the strings in the `SShift` constant.

For an example, see PollShiftStateEvent (660).

Errors: None.

See also: FunctionKeyName (653), KeyEventToString (659)

15.6.24 TranslateKeyEvent

Synopsis: Translate raw event to ascii key event

Declaration:
```
function TranslateKeyEvent(KeyEvent: TKeyEvent) : TKeyEvent
```

Visibility: default

Description: `TranslateKeyEvent` performs ASCII translation of the `KeyEvent`. It translates a physical key to a function key if the key is a function key, and translates the physical key to the ordinal of the ascii character if there is an equivalent character key.

For an example, see GetKeyEvent (654)

Errors: None.

See also: TranslateKeyEventUniCode (663)

15.6.25 TranslateKeyEventUniCode

Synopsis: Translate raw event to UNICode key event

Declaration:
```
function TranslateKeyEventUniCode(KeyEvent: TKeyEvent) : TKeyEvent
```

Visibility: default

Description: `TranslateKeyEventUniCode` performs Unicode translation of the `KeyEvent`. It is not yet implemented for all platforms.

Errors: If the function is not yet implemented, then the `ErrorCode` of the system unit will be set to `errKbdNotImplemented`

See also: TranslateKeyEvent (663)

Table 15.1: Key Scancodes

Code	Key	Code	Key	Code	Key
00	NoKey	3D	F3	70	ALT-F9
01	ALT-Esc	3E	F4	71	ALT-F10
02	ALT-Space	3F	F5	72	CTRL-PrtSc
04	CTRL-Ins	40	F6	73	CTRL-Left
05	SHIFT-Ins	41	F7	74	CTRL-Right
06	CTRL-Del	42	F8	75	CTRL-end
07	SHIFT-Del	43	F9	76	CTRL-PgDn
08	ALT-Back	44	F10	77	CTRL-Home
09	ALT-SHIFT-Back	47	Home	78	ALT-1
0F	SHIFT-Tab	48	Up	79	ALT-2
10	ALT-Q	49	PgUp	7A	ALT-3
11	ALT-W	4B	Left	7B	ALT-4
12	ALT-E	4C	Center	7C	ALT-5
13	ALT-R	4D	Right	7D	ALT-6
14	ALT-T	4E	ALT-GrayPlus	7E	ALT-7
15	ALT-Y	4F	end	7F	ALT-8
16	ALT-U	50	Down	80	ALT-9
17	ALT-I	51	PgDn	81	ALT-0
18	ALT-O	52	Ins	82	ALT-Minus
19	ALT-P	53	Del	83	ALT-Equal
1A	ALT-LftBrack	54	SHIFT-F1	84	CTRL-PgUp
1B	ALT-RgtBrack	55	SHIFT-F2	85	F11
1E	ALT-A	56	SHIFT-F3	86	F12
1F	ALT-S	57	SHIFT-F4	87	SHIFT-F11
20	ALT-D	58	SHIFT-F5	88	SHIFT-F12
21	ALT-F	59	SHIFT-F6	89	CTRL-F11
22	ALT-G	5A	SHIFT-F7	8A	CTRL-F12
23	ALT-H	5B	SHIFT-F8	8B	ALT-F11
24	ALT-J	5C	SHIFT-F9	8C	ALT-F12
25	ALT-K	5D	SHIFT-F10	8D	CTRL-Up
26	ALT-L	5E	CTRL-F1	8E	CTRL-Minus
27	ALT-SemiCol	5F	CTRL-F2	8F	CTRL-Center
28	ALT-Quote	60	CTRL-F3	90	CTRL-GreyPlus
29	ALT-OpQuote	61	CTRL-F4	91	CTRL-Down
2B	ALT-BkSlash	62	CTRL-F5	94	CTRL-Tab
2C	ALT-Z	63	CTRL-F6	97	ALT-Home
2D	ALT-X	64	CTRL-F7	98	ALT-Up
2E	ALT-C	65	CTRL-F8	99	ALT-PgUp
2F	ALT-V	66	CTRL-F9	9B	ALT-Left
30	ALT-B	67	CTRL-F10	9D	ALT-Right
31	ALT-N	68	ALT-F1	9F	ALT-end
32	ALT-M	69	ALT-F2	A0	ALT-Down
33	ALT-Comma	6A	ALT-F3	A1	ALT-PgDn
34	ALT-Period	6B	ALT-F4	A2	ALT-Ins
35	ALT-Slash	6C	ALT-F5	A3	ALT-Del
37	ALT-GreyAst	6D	ALT-F6	A5	ALT-Tab
3B	F1	6E	ALT-F7		
3C	F2	6F	ALT-F8		

Table 15.2: Special keys scan codes

Key	Code	SHIFT-Key	CTRL-Key	Alt-Key
NoKey	00			
F1	3B	54	5E	68
F2	3C	55	5F	69
F3	3D	56	60	6A
F4	3E	57	61	6B
F5	3F	58	62	6C
F6	40	59	63	6D
F7	41	5A	64	6E
F8	42	5A	65	6F
F9	43	5B	66	70
F10	44	5C	67	71
F11	85	87	89	8B
F12	86	88	8A	8C
Home	47		77	97
Up	48		8D	98
PgUp	49		84	99
Left	4B		73	9B
Center	4C		8F	
Right	4D		74	9D
end	4F		75	9F
Down	50		91	A0
PgDn	51		76	A1
Ins	52	05	04	A2
Del	53	07	06	A3
Tab	8	0F	94	A5
GreyPlus			90	4E

Table 15.3: Structure of TKeyRecord

Field	Meaning
KeyCode	Depending on `flags` either the physical representation of a key (under DOS scancode, ascii code pair), or the t
ShiftState	Shift-state when this key was pressed (or shortly after)
Flags	Determine how to interpret `KeyCode`

Chapter 16

Reference for unit 'Linux'

16.1 Used units

Table 16.1: Used units by unit 'Linux'

Name	Page
BaseUnix	94

16.2 Overview

The linux unit contains linux specific operating system calls.

The platform independent functionality of the FPC 1.0.X version of the linux unit has been split out over the unix (1538), baseunix (94) and unixutil (1589) units.

The X86-specific parts have been moved to the X86 (1619) unit.

People wanting to use the old version (FPC 1.0.X and before) of the linux can use the oldlinux (938) unit instead.

16.3 Constants, types and variables

16.3.1 Constants

```
CAP_AUDIT_CONTROL = 30
```

Allow manipulation of kernel auditing features

```
CAP_AUDIT_WRITE = 29
```

Allow writing to kernel audit log

```
CAP_CHOWN = 0
```

Perform chown operation

```
CAP_DAC_OVERRIDE = 1
```

Bypass file operation (rwx) checks

```
CAP_DAC_READ_SEARCH = 2
```

Bypass file read-only operation checks

```
CAP_FOWNER = 3
```

Bypass owner ID checks

```
CAP_FSETID = 4
```

Do not clear SUID/GUID bits on modified files

```
CAP_FS_MASK = $1f
```

?

```
CAP_IPC_LOCK = 14
```

Allow memory locking calls

```
CAP_IPC_OWNER = 15
```

Bypass permission checks on IPC operations

```
CAP_KILL = 5
```

Bypass permission checks for sending signals

```
CAP_LEASE = 28
```

Allow file leases

```
CAP_LINUX_IMMUTABLE = 9
```

Allow setting ext2 file attributes

```
CAP_MKNOD = 27
```

Allow creation of special files through mknod calls

```
CAP_NET_ADMIN = 12
```

Allow network operations (e.g. setting socket options)

```
CAP_NET_BIND_SERVICE = 10
```

Allow binding to ports less than 1024

`CAP_NET_BROADCAST = 11`

Allow socket broadcast operations

`CAP_NET_RAW = 13`

Allow use of RAW and PACKET sockets

`CAP_SETGID = 6`

Allow GID manipulations

`CAP_SETPCAP = 8`

Allow to set other process' capabilities

`CAP_SETUID = 7`

Allow process ID manipulations

`CAP_SYS_ADMIN = 21`

Allow various system administration calls

`CAP_SYS_BOOT = 22`

Allow reboot calls

`CAP_SYS_CHROOT = 18`

Allow chroot calls.

`CAP_SYS_MODULE = 16`

Allow loading/unloading of kernel modules

`CAP_SYS_NICE = 23`

Allowing raising process and thread priorities

`CAP_SYS_PACCT = 20`

Allow acct calls

`CAP_SYS_PTRACE = 19`

Allow ptrace calls

`CAP_SYS_RAWIO = 17`

Allow raw I/O port operations

```
CAP_SYS_RESOURCE = 24
```

Allow use of special resources or raising of resource limits

```
CAP_SYS_TIME = 25
```

Allow system or real-time clock modification

```
CAP_SYS_TTY_CONFIG = 26
```

Allow vhangup calls

```
CLONE_CHILD_CLEARTID = $00200000
```

Clone option: Erase child thread ID in child memory space when child exits.

```
CLONE_CHILD_SETTID = $01000000
```

Clone option: Store child thread ID in child memory.

```
CLONE_DETACHED = $00400000
```

Clone option: Start clone detached.

```
CLONE_FILES = $00000400
```

Clone (666) option: open files shared between processes

```
CLONE_FS = $00000200
```

Clone (666) option: fs info shared between processes

```
CLONE_NEWNS = $00020000
```

Clone options: Start child in new (filesystem) namespace.

```
CLONE_PARENT = $00008000
```

Clone options: Set child parent to parent of calling process.

```
CLONE_PARENT_SETTID = $00100000
```

Clone option: Store child thread ID in memory in both parent and child.

```
CLONE_PID = $00001000
```

Clone (666) option: PID shared between processes

```
CLONE_PTRACE = $00002000
```

Clone options: if parent is traced, trace child also

```
CLONE_SETTLS = $00080000
```

Clone option: The newtls parameter is the TLS descriptor of the child

```
CLONE_SIGHAND = $00000800
```

Clone (666) option: signal handlers shared between processes

```
CLONE_STOPPED = $02000000
```

Clone option: Start child in stopped state.

```
CLONE_SYSVSEM = $00040000
```

Clone option: Caller and child share the same semaphore undo values

```
CLONE_THREAD = $00010000
```

Clone options: Set child in thread group of calling process.

```
CLONE_UNTRACED = $00800000
```

Clone option: Do not allow a ptrace call on this clone.

```
CLONE_VFORK = $00004000
```

Clone options: suspend parent till child execs

```
CLONE_VM = $00000100
```

Clone (666) option: VM shared between processes

```
CSIGNAL = $000000ff
```

Clone (666) option: Signal mask to be sent at exit

```
EPOLLERR = $08
```

Poll error condition

```
EPOLLET = $80000000
```

Undocumented

```
EPOLLHUP = $10
```

Poll hung up

```
EPOLLIN = $01
```

Poll input file descriptor ready event

```
EPOLLOUT = $04
```

Poll output file descriptor ready event

```
EPOLLPRI = $02
```

Priority data available on input file descriptor

```
EPOLL_CTL_ADD = 1
```

Add filedescriptor to list of events

```
EPOLL_CTL_DEL = 2
```

Delete event for filedescriptor

```
EPOLL_CTL_MOD = 3
```

Modify event for filedescriptor

```
FUTEX_CMP_REQUEUE = 4
```

Futex option: requeue waiting processes on other futex, but check it's value first

```
FUTEX_FD = 2
```

Futex option: Associate file descriptor with futex.

```
FUTEX_LOCK_PI = 6
```

Futex option: Undocumented

```
FUTEX_OP_ADD = 1
```

Futex operation: Undocumented

```
FUTEX_OP_ANDN = 3
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_EQ = 0
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_GE = 5
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_GT = 4
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_LE = 3
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_LT = 2
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_NE = 1
```

Futex operation: Undocumented

```
FUTEX_OP_OPARG_SHIFT = 8
```

Futex operation: Undocumented

```
FUTEX_OP_OR = 2
```

Futex operation: Undocumented

```
FUTEX_OP_SET = 0
```

Futex operation: Undocumented

```
FUTEX_OP_XOR = 4
```

Futex operation: Undocumented

```
FUTEX_REQUEUE = 3
```

Futex option: requeue waiting processes on other futex.

```
FUTEX_TRYLOCK_PI = 8
```

Futex option: Undocumented

```
FUTEX_UNLOCK_PI = 7
```

Futex option: Undocumented

```
FUTEX_WAIT = 0
```

Futex option: Wait on futex till wake call arrives.

```
FUTEX_WAKE = 1
```

Futex option: wakes any waiting processes on this futex

```
FUTEX_WAKE_OP = 5
```

Futex option: Undocumented

```
GIO_CMAP = $4B70
```

IOCTL: Get colour palette on VGA+

```
GIO_FONT = $4B60
```

IOCTL: Get font in expanded form.

```
GIO_FONTX = $4B6B
```

IOCTL: Get font in `consolefontdesc` record.

```
GIO_SCRNMAP = $4B40
```

IOCTL: get screen mapping from kernel

```
GIO_UNIMAP = $4B66
```

IOCTL: get unicode-to-font mapping from kernel

```
GIO_UNISCRNMAP = $4B69
```

IOCTL: get full Unicode screen mapping

```
KB_101 = 2
```

IOCTL: Keyboard types: 101 keys

```
KB_84 = 1
```

IOCTL: Keyboard types: 84 keys

```
KB_OTHER = 3
```

IOCTL: Keyboard types: other type

```
KDADDIO = $4B34
```

IOCTL: add i/o port as valid

```
KDDELIO = $4B35
```

IOCTL: delete i/o port as valid

```
KDDISABIO = $4B37
```

IOCTL: disable i/o to video board

```
KDENABIO = $4B36
```

IOCTL: enable i/o to video board

```
KDFONTOP = $4B72
```

IOCTL: font operations

```
KDGETKEYCODE = $4B4C
```

IOCTL: read kernel keycode table entry

```
KDGETLED = $4B31
```

IOCTL: return current led state

```
KDGETMODE = $4B3B
```

IOCTL: get current mode

```
KDGKBDIACR = $4B4A
```

IOCTL: read kernel accent table

```
KDGKBTYPE = $4B33
```

IOCTL: get keyboard type

```
KDMAPDISP = $4B3C
```

IOCTL: map display into address space

```
KDMKTONE = $4B30
```

IOCTL: generate tone

```
KDSETKEYCODE = $4B4D
```

IOCTL: write kernel keycode table entry

```
KDSETLED = $4B32
```

IOCTL: set led state

```
KDSETMODE = $4B3A
```

IOCTL: set text/graphics mode

```
KDSIGACCEPT = $4B4E
```

IOCTL: accept kbd generated signals

```
KDSKBDIACR = $4B4B
```

IOCTL: write kernel accent table

```
KDUNMAPDISP = $4B3D
```

IOCTL: unmap display from address space

```
KD_GRAPHICS = 1
```

IOCTL: Tty modes: graphics mode

```
KD_TEXT = 0
```

IOCTL: Tty modes: Text mode

```
KD_TEXT0 = 2
```

IOCTL: Tty modes: Text mode (obsolete)

```
KD_TEXT1 = 3
```

IOCTL: Tty modes: Text mode (obsolete)

```
KIOCSOUND = $4B2F
```

IOCTL: start/stop sound generation (0 for off)

```
LED_CAP = 4
```

IOCTL: LED_CAP : caps lock led

```
LED_NUM = 2
```

IOCTL: LED_SCR : Num lock led

```
LED_SCR = 1
```

IOCTL: LED_SCR : scroll lock led

```
LINUX_CAPABILITY_VERSION = $19980330
```

Current capability version in use by kernel

```
MAP_DENYWRITE = $800
```

Read-only

```
MAP_EXECUTABLE = $1000
```

Memory area is marked as executable

```
MAP_GROWSDOWN = $100
```

Memory map grows down, like stack

```
MAP_LOCKED = $2000
```

Memory pages are locked

```
MAP_NORESERVE = $4000
```

Do not check for reservations

```
MODIFY_LDT_CONTENTS_CODE = 2
```

Modify_ldt option: Undocumented

```
MODIFY_LDT_CONTENTS_DATA = 0
```

Modify_ldt option: Undocumented

```
MODIFY_LDT_CONTENTS_STACK = 1
```

Modify_ldt option: Undocumented

```
PIO_CMAP = $4B71
```

IOCTL: Set colour palette on VGA+

```
PIO_FONT = $4B61
```

IOCTL: Use font in expanded form.

```
PIO_FONTRESET = $4B6D
```

IOCTL: Reset to default font

```
PIO_FONTX = $4B6C
```

IOCTL: Set font in `consolefontdesc` record.

```
PIO_SCRNMAP = $4B41
```

IOCTL: put screen mapping table in kernel

```
PIO_UNIMAP = $4B67
```

IOCTL: put unicode-to-font mapping in kernel

```
PIO_UNIMAPCLR = $4B68
```

IOCTL: clear table, possibly advise hash algorithm

```
PIO_UNISCRNMAP = $4B6A
```

IOCTL: set full Unicode screen mapping

```
SPLICE_F_GIFT = 8
```

Pages spliced in are a gift

```
SPLICE_F_MORE = 4
```

Expect more data

```
SPLICE_F_MOVE = 1
```

Move pages instead of copying

```
SPLICE_F_NONBLOCK = 2
```

Don't block on pipe splicing operations

```
UD_CONTENTS_CODE = MODIFY_LDT_CONTENTS_CODE shl 1
```

TLS segment descriptor: Undocumented

```
UD_CONTENTS_DATA = MODIFY_LDT_CONTENTS_DATA shl 1
```

TLS segment descriptor: Undocumented

```
UD_CONTENTS_STACK = MODIFY_LDT_CONTENTS_STACK shl 1
```

TLS segment descriptor: Undocumented

```
UD_LIMIT_IN_PAGES = $10
```

TLS segment descriptor: Undocumented

```
UD_LM = $80
```

TLS segment descriptor: Undocumented

```
UD_READ_EXEC_ONLY = $08
```

TLS segment descriptor: Undocumented

```
UD_SEG_32BIT = $01
```

TLS segment descriptor : Undocumented

```
UD_SEG_NOT_PRESENT = $20
```

TLS segment descriptor: Undocumented

```
UD_USEABLE = $40
```

TLS segment descriptor: Undocumented

16.3.2 Types

```
EPoll_Data = record
end
```

Data structure used in EPOLL IOCTL call.

```
EPoll_Event = record
  Events : cuint32;
  Data : TEPoll_Data;
end
```

Structure used in epoll_ctl (681) call.

```
PEPoll_Data = ^EPoll_Data
```

Pointer to EPoll_Data (678) record

```
PEpoll_Event = ^EPoll_Event
```

Pointer to EPoll_Event (678) type

```
PSysInfo = ^TSysInfo
```

Pointer to TSysInfo (679) record.

```
Puser_cap_data = ^user_cap_data
```

Pointer to user_cap_data (679) record

```
Puser_cap_header = ^user_cap_header
```

Pointer to user_cap_header (679) record

```
PUser_Desc = ^user_desc
```

`PUser_Desc` is a pointer to the user_desc (679) type.

```
TCloneFunc = function(args: pointer) : LongInt
```

Clone function prototype.

```
TEPoll_Data = EPoll_Data
```

Alias for EPoll_Data (678) type

```
TEPoll_Event = EPoll_Event
```

Alias for EPoll_Event (678) type

```
TSysInfo = record
  uptime : clong;
  loads : Array[0..2] of culong;
  totalram : culong;
  freeram : culong;
  sharedram : culong;
  bufferram : culong;
  totalswap : culong;
  freeswap : culong;
  procs : cushort;
  pad : cushort;
  totalhigh : culong;
  freehigh : culong;
  mem_unit : cuint;
  _f : Array[0..19-2*sizeof(clong)-sizeof(cint)] of cchar;
end
```

Record with system information, used by the SysInfo (682) call.

```
TUser_Desc = user_desc
```

`TUser_Desc` is an alias for the user_desc (679) type.

```
user_cap_data = packed record
  effective : cardinal;
  permitted : cardinal;
  inheritable : cardinal;
end
```

`user_cap_data` describes the set of capabilities for the indicated thread.

```
user_cap_header = packed record
  version : cardinal;
  pid : cardinal;
end
```

`user_cap_header` describes the root user capabilities for the current thread, as set by getcap (666) and setcap (666)

```
user_desc = packed record
  entry_number : cint;
  base_addr : cuint;
  limit : cuint;
  flags : cuint;
end
```

`user_desc` is the TLS (Thread Local Storage) segment descriptor used in the `Clone` call. It should not be used, as it contains highly kernel-specific data.

16.4 Procedures and functions

16.4.1 capget

Synopsis: Return the capabilities for the indicated thread

Declaration: `function capget(header: Puser_cap_header;data: Puser_cap_data) : cint`

Visibility: default

Description: `capget` returns the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and fperrno is set to the error.

See also: capset (680)

16.4.2 capset

Synopsis: Set the capabilities for the indicated thread

Declaration: `function capset(header: Puser_cap_header;data: Puser_cap_data) : cint`

Visibility: default

Description: `capget` sets the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and fperrno is set to the error.

See also: capget (680)

16.4.3 epoll_create

Synopsis: Create new epoll file descriptor

Declaration: `function epoll_create(size: cint) : cint`

Visibility: default

Description: `epoll_create` creates a new epoll file descriptor. The `size` argument indicates to the kernel approximately how many structures should be allocated, but is by no means an upper limit.

On success, a file descriptor is returned that can be used in subsequent epoll_ctl (681) or epoll_wait (681) calls, and should be closed using the fpClose (136) call.

Errors: On error, -1 is returned, and errno (94) is set.

See also: epoll_ctl (681), epoll_wait (681), #rtl.baseunix.fpClose (136)

16.4.4 epoll_ctl

Synopsis: Modify an epoll file descriptor

Declaration:
```
function epoll_ctl(epfd: cint;op: cint;fd: cint;event: PEpoll_Event)
                        : cint
```

Visibility: default

Description: `epoll_ctl` performs the `op` operation on epoll file descriptor `epfd`. The operation will be monitored on file descriptor `fd`, and is optionally controlled by `event`.

`op` can be one of the following values:

EPOLL_CTL_ADD Add filedescriptor to list of events

EPOLL_CTL_MOD Modify event for filedescriptor

EPOLL_CTL_DEL Delete event for filedescriptor

The `events` field in `event_data` is a bitmask of one or more of the following values:

EPOLLIN The file is ready for read operations

EPOLLOUT The file is ready for write operations.

EPOLLPRI Urgent data is available for read operations.

EPOLLERR An error condition is signaled on the file descriptor.

EPOLLHUP A Hang up happened on the file descriptor.

EPOLLET Set the Edge Triggered behaviour for the file descriptor.

EPOLLONESHOT Set One-Shot behaviour for the file descriptor. The event will be triggered only once.

Errors: On error -1 is returned, and errno is set accordingly.

See also: epoll_create (680), epoll_wait (681), #rtl.baseunix.fpClose (136)

16.4.5 epoll_wait

Synopsis: Wait for an event on an epoll file descriptor.

Declaration:
```
function epoll_wait(epfd: cint;events: PEpoll_Event;maxevents: cint;
                        timeout: cint) : cint
```

Visibility: default

Description: `epoll_wait` waits for `timeout` milliseconds for an event to occur on epoll file descriptor `epfd`. If `timeout` is -1, it waits indefinitely, if `timeour` is zero, it does not wait, but returns immediatly, even if no events were detected.

On return, data for at most `maxevents` will be returned in the memory pointed to by `events`. The function returns the number of file descriptors for which events were reported. This can be zero if the timeout was reached.

Errors: On error -1 is returned, and errno is set accordingly.

See also: epoll_create (680), epoll_ctl (681), #rtl.baseunix.fpClose (136)

16.4.6 futex_op

Synopsis: Futex operation:

Declaration: `function futex_op(op: cint;oparg: cint;cmp: cint;cmparg: cint) : cint`

Visibility: default

Description: `FUTEX_OP` Performs an operation on a futex:

```
FUTEX_OP := ((op and $F) shl 28) or
            ((cmp and $F) shl 24) or
            ((oparg and $FFF) shl 12)
            or (cmparg and $FFF);
```

16.4.7 Sysinfo

Synopsis: Return kernel system information

Declaration: `function Sysinfo(Info: PSysInfo) : cint`

Visibility: default

Description: `SysInfo` returns system information in `Info`. Returned information in `Info` includes:

uptimeNumber of seconds since boot.

loads1, 5 and 15 minute load averages.

totalramtotal amount of main memory.

freeramamount of free memory.

sharedramamount of shared memory.

bufferramamount of memory used by buffers.

totalswaptotal amount of swapspace.

freeswapamount of free swapspace.

procsnumber of current processes.

Errors: None.

See also: #rtl.baseunix.fpUname (179)

Listing: ./linuxex/ex64.pp

```
program Example64;

{ Example to demonstrate the SysInfo function.
  Sysinfo is Linux-only. }

{$ifdef Linux}
Uses Linux;

Function Mb(L : Longint) : longint;

begin
  Mb:=L div (1024*1024);
end;

Var Info : TSysInfo;
```

```
    D,M,Secs,H : longint;
{$endif}

begin
  {$ifdef Linux}
  If Not SysInfo(Info) then
    Halt(1);
  With Info do
    begin
    D:=Uptime div (3600*24);
    UpTime:=UpTime mod (3600*24);
    h:=uptime div 3600;
    uptime:=uptime mod 3600;
    m:=uptime div 60;
    secs:=uptime mod 60;
    Writeln('Uptime : ',d,'days, ',h,' hours, ',m,' min, ',secs,' s.');
    Writeln('Loads  : ',Loads[1],'/',Loads[2],'/',Loads[3]);
    Writeln('Total Ram  : ',Mb(totalram),'Mb.');
    Writeln('Free Ram   : ',Mb(freeram),'Mb.');
    Writeln('Shared Ram : ',Mb(sharedram),'Mb.');
    Writeln('Buffer Ram : ',Mb(bufferram),'Mb.');
    Writeln('Total Swap : ',Mb(totalswap),'Mb.');
    Writeln('Free Swap  : ',Mb(freeswap),'Mb.');
    end;
  {$endif}
end.
```

Chapter 17

Reference for unit 'math'

17.1 Geometrical functions

Table 17.1:

Name	Description
hypot (701)	Hypotenuse of triangle
norm (713)	Euclidian norm

17.2 Statistical functions

Table 17.2:

Name	Description
mean (709)	Mean of values
meanandstddev (709)	Mean and standard deviation of values
momentskewkurtosis (712)	Moments, skew and kurtosis
popnstddev (714)	Population standarddeviation
popnvariance (715)	Population variance
randg (717)	Gaussian distributed randum value
stddev (722)	Standard deviation
sum (722)	Sum of values
sumofsquares (723)	Sum of squared values
sumsandsquares (724)	Sum of values and squared values
totalvariance (726)	Total variance of values
variance (727)	variance of values

Table 17.3:

Name	Description
ceil (694)	Round to infinity
floor (699)	Round to minus infinity
frexp (699)	Return mantissa and exponent

17.3 Number converting

17.4 Exponential and logarithmic functions

Table 17.4:

Name	Description
intpower (702)	Raise float to integer power
ldexp (704)	Calculate $2p̂ x$
lnxp1 (704)	calculate `log(x+1)`
log10 (705)	calculate 10-base log
log2 (705)	calculate 2-base log
logn (706)	calculate N-base log
power (715)	raise float to arbitrary power

17.5 Hyperbolic functions

Table 17.5:

Name	Description
arcosh (691)	caculate reverse hyperbolic cosine
arsinh (693)	caculate reverse hyperbolic sine
artanh (693)	caculate reverse hyperbolic tangent
cosh (695)	calculate hyperbolic cosine
sinh (721)	calculate hyperbolic sine
tanh (725)	calculate hyperbolic tangent

17.6 Trigoniometric functions

17.7 Angle unit conversion

Routines to convert angles between different angle units.

Table 17.6:

Name	Description
arccos (690)	calculate reverse cosine
arcsin (691)	calculate reverse sine
arctan2 (692)	calculate reverse tangent
cotan (696)	calculate cotangent
sincos (720)	calculate sine and cosine
tan (725)	calculate tangent

Table 17.7:

Name	Description
cycletorad (697)	convert cycles to radians
degtograd (697)	convert degrees to grads
degtorad (698)	convert degrees to radians
gradtodeg (700)	convert grads to degrees
gradtorad (701)	convert grads to radians
radtocycle (716)	convert radians to cycles
radtodeg (716)	convert radians to degrees
radtograd (717)	convert radians to grads

17.8 Min/max determination

Functions to determine the minimum or maximum of numbers:

Table 17.8:

Name	Description
max (706)	Maximum of 2 values
maxIntValue (707)	Maximum of an array of integer values
maxvalue (708)	Maximum of an array of values
min (710)	Minimum of 2 values
minIntValue (711)	Minimum of an array of integer values
minvalue (711)	Minimum of an array of values

17.9 Used units

17.10 Overview

This document describes the math unit. The `math` unit was initially written by Florian Klaempfl. It provides mathematical functions which aren't covered by the system unit.

This chapter starts out with a definition of all types and constants that are defined, after which an overview is presented of the available functions, grouped by category, and the last part contains a complete explanation of each function.

The following things must be taken into account when using this unit:

Table 17.9: Used units by unit 'math'

Name	Page
sysutils	1350

1. This unit is compiled in Object Pascal mode so all `integers` are 32 bit.
2. Some overloaded functions exist for data arrays of integers and floats. When using the address operator (`@`) to pass an array of data to such a function, make sure the address is typecasted to the right type, or turn on the 'typed address operator' feature. failing to do so, will cause the compiler not be able to decide which function you want to call.

17.11 Constants, types and variables

17.11.1 Constants

```
EqualsValue = 0
```

Values are the same

```
GreaterThanValue = High ( TValueRelationship )
```

First values is greater than second value

```
Infinity = 1.0 / 0.0
```

Value is infinity

```
LessThanValue = Low ( TValueRelationship )
```

First value is less than second value

```
MaxExtended = 1.1e + 4932
```

Maximum value of extended type

```
MaxFloat = MaxExtended
```

Maximum value of float type

```
MinExtended = 3.4e - 4932
```

Minimum value (closest to zero) of extended type

```
MinFloat = MinExtended
```

Minimum value (closest to zero) of float type

```
NaN = 0.0 / 0.0
```

Value is Not a Number

```
NegativeValue = Low ( TValueSign )
```

Value is negative

```
NegInfinity = -1.0 / 0.0
```

Value is negative (minus) infinity

```
PositiveValue = High ( TValueSign )
```

Value is positive

```
ZeroValue = 0
```

Value is zero

17.11.2 Types

```
float = extended
```

All calculations are done with the Float type. This allows to recompile the unit with a different float type to obtain a desired precision. The pointer type PFloat (688) is used in functions that accept an array of values of arbitrary length.

```
PFloat = ^float
```

Pointer to Float (688) type.

```
PInteger = ObjPas.PInteger
```

Pointer to integer type

```
TFPUException = (exInvalidOp,exDenormalized,exZeroDivide,exOverflow,
                 exUnderflow,exPrecision)
```

Table 17.10: Enumeration values for type TFPUException

Value	Explanation
exDenormalized	
exInvalidOp	Invalid operation error
exOverflow	Float overflow error
exPrecision	Precision error
exUnderflow	Float underflow error
exZeroDivide	Division by zero error.

Type describing Floating Point processor exceptions.

Table 17.11: Enumeration values for type TFPUPrecisionMode

Value	Explanation
pmDouble	Double-type precision
pmExtended	Extended-type precision
pmReserved	?
pmSingle	Single-type precision

```
TFPUExceptionMask= Set of (exDenormalized,exInvalidOp,exOverflow,
                           exPrecision,exUnderflow,exZeroDivide)
```

Type to set the Floating Point Unit exception mask.

```
TFPUPrecisionMode = (pmSingle,pmReserved,pmDouble,pmExtended)
```

Type describing the default precision for the Floating Point processor.

```
TFPURoundingMode = (rmNearest,rmDown,rmUp,rmTruncate)
```

Table 17.12: Enumeration values for type TFPURoundingMode

Value	Explanation
rmDown	Round to biggest integer smaller than value.
rmNearest	Round to nearest integer value
rmTruncate	Cut off fractional part.
rmUp	Round to smallest integer larger than value.

Type describing the rounding mode for the Floating Point processor.

```
tpaymenttime = (ptendofperiod,ptstartofperiod)
```

Table 17.13: Enumeration values for type tpaymenttime

Value	Explanation
ptendofperiod	End of period.
ptstartofperiod	Start of period.

Type used in financial (interest) calculations.

```
TRoundToRange = -37..37
```

`TRoundToRange` is the range of valid digits to be used in the RoundTo (718) function.

```
TValueRelationship = -1..1
```

Type to describe relational order between values

```
TValueSign = -1..1
```

Type indicating sign of a valuea

17.12 Procedures and functions

17.12.1 arccos

Synopsis: Return inverse cosine

Declaration: `function arccos(x: float) : float`

Visibility: default

Description: `Arccos` returns the inverse cosine of its argument `x`. The argument `x` should lie between -1 and 1 (borders included).

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: arcsin (691), arcosh (691), arsinh (693), artanh (693)

Listing: ./mathex/ex1.pp

```
Program Example1;

{ Program to demonstrate the arccos function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    WriteIn(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arccos(1));
  WriteRadDeg (arccos(sqrt(3)/2));
  WriteRadDeg (arccos(sqrt(2)/2));
  WriteRadDeg (arccos(1/2));
  WriteRadDeg (arccos(0));
  WriteRadDeg (arccos(-1));
end.
```

17.12.2 arccosh

Synopsis: Return inverse hyperbolic cosine

Declaration: `function arccosh(x: float) : float`

Visibility: default

Description: `arccosh` returns the inverse hyperbolic cosine of it's argument `x`.

This function is an alias for arcosh (691), provided for Delphi compatibility.

See also: arcosh (691)

17.12.3 arcosh

Synopsis: Return inverse hyperbolic cosine

Declaration: `function arcosh(x: float) : float`

Visibility: default

Description: `Arcosh` returns the inverse hyperbolic cosine of its argument `x`. The argument `x` should be larger than 1. The `arccosh` variant of this function is supplied for Delphi compatibility.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: cosh (695), sinh (721), arcsin (691), arsinh (693), artanh (693), tanh (725)

Listing: ./mathex/ex3.pp

```
Program Example3;

{ Program to demonstrate the arcosh function. }

Uses math;

begin
  Writeln(arcosh(1));
  Writeln(arcosh(2));
end.
```

17.12.4 arcsin

Synopsis: Return inverse sine

Declaration: `function arcsin(x: float) : float`

Visibility: default

Description: `Arcsin` returns the inverse sine of its argument `x`. The argument `x` should lie between -1 and 1.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: arccos (690), arcosh (691), arsinh (693), artanh (693)

Listing: ./mathex/ex2.pp

```
Program Example1;

{ Program to demonstrate the arcsin function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    Writeln(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arcsin(1));
  WriteRadDeg (arcsin(sqrt(3)/2));
```

```
  WriteRadDeg (arcsin(sqrt(2)/2));
  WriteRadDeg (arcsin(1/2));
  WriteRadDeg (arcsin(0));
  WriteRadDeg (arcsin(-1));
end.
```

17.12.5 arcsinh

Synopsis: Return inverse hyperbolic sine

Declaration: `function arcsinh(x: float) : float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic sine of it's argument `x`.

This function is an alias for arsinh (693), provided for Delphi compatibility.

See also: arsinh (693)

17.12.6 arctan2

Synopsis: Return arctangent of (y/x)

Declaration: `function arctan2(y: float;x: float) : float`

Visibility: default

Description: `arctan2` calculates `arctan(y/x)`, and returns an angle in the correct quadrant. The returned angle will be in the range $-\pi$ to π radians. The values of `x` and `y` must be between -2\^{}64 and 2\^{}64, moreover `x` should be different from zero. On Intel systems this function is implemented with the native intel `fpatan` instruction.

Errors: If `x` is zero, an overflow error will occur.

See also: arccos (690), arcosh (691), arsinh (693), artanh (693)

Listing: ./mathex/ex6.pp

```
Program Example6;

{ Program to demonstrate the arctan2 function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    WriteIn(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arctan2(1,1));
end.
```

17.12.7 arctanh

Synopsis: Return inverse hyperbolic tangent

Declaration: `function arctanh(x: float) : float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic tangent of it's argument `x`.

This function is an alias for artanh (693), provided for Delphi compatibility.

See also: artanh (693)

17.12.8 arsinh

Synopsis: Return inverse hyperbolic sine

Declaration: `function arsinh(x: float) : float`

Visibility: default

Description: `arsinh` returns the inverse hyperbolic sine of its argument `x`. The `arscsinh` variant of this function is supplied for Delphi compatibility.

Errors: None.

See also: arcosh (691), arccos (690), arcsin (691), artanh (693)

Listing: ./mathex/ex4.pp

```
Program Example4;

{ Program to demonstrate the arsinh function. }

Uses math;

begin
  WriteLn(arsinh(0));
  WriteLn(arsinh(1));
end.
```

17.12.9 artanh

Synopsis: Return inverse hyperbolic tangent

Declaration: `function artanh(x: float) : float`

Visibility: default

Description: `artanh` returns the inverse hyperbolic tangent of its argument `x`, where `x` should lie in the interval [-1,1], borders included. The `arctanh` variant of this function is supplied for Delphi compatibility.

Errors: In case `x` is not in the interval [-1,1], an `EInvalidArgument` exception is raised.

See also: arcosh (691), arccos (690), arcsin (691), artanh (693)

Listing: ./mathex/ex5.pp

```
Program Example5;

{ Program to demonstrate the artanh function. }

Uses math;

begin
  WriteLn(artanh(0));
  WriteLn(artanh(0.5));
end.
```

17.12.10 ceil

Synopsis: Return the lowest integer number greater than or equal to argument

Declaration: `function ceil(x: float) : Integer`

Visibility: default

Description: `Ceil` returns the lowest integer number greater than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If the asolute value of `x` is larger than maxint, an overflow error will occur.

See also: floor (699)

Listing: ./mathex/ex7.pp

```
Program Example7;

{ Program to demonstrate the Ceil function. }

Uses math;

begin
  WriteLn(Ceil(-3.7)); // should be -3
  WriteLn(Ceil(3.7));  // should be 4
  WriteLn(Ceil(-4.0)); // should be -4
end.
```

17.12.11 ClearExceptions

Synopsis: Clear Floating Point Unit exceptions

Declaration: `procedure ClearExceptions(RaisePending: Boolean)`

Visibility: default

Description: Clear Floating Point Unit exceptions

17.12.12 CompareValue

Synopsis: Compare 2 values

Declaration:
```
function CompareValue(const A: Integer;const B: Integer)
                        : TValueRelationship
function CompareValue(const A: Int64;const B: Int64)
                        : TValueRelationship
function CompareValue(const A: QWord;const B: QWord)
                        : TValueRelationship
function CompareValue(const A: Extended;const B: Extended;
                      delta: Extended) : TValueRelationship
```

Visibility: default

Description: `CompareValue` compares 2 integer or floating point values `A` and `B` and returns one of the following values:

-1if `A<B`
0if `A=B`
-1if `A>B`

See also: TValueRelationship (689)

17.12.13 cosecant

Synopsis: Calculate cosecant

Declaration:
```
function cosecant(x: float) : float
```

Visibility: default

Description: `cosecant` calculates the cosecant (1/`sin(x)`) of its argument `x`.

Errors: If 0 or 180 degrees is specified, an exception will be raised.

See also: secant (719)

17.12.14 cosh

Synopsis: Return hyperbolic cosine

Declaration:
```
function cosh(x: float) : float
```

Visibility: default

Description: `Cosh` returns the hyperbolic cosine of it's argument {x}.

Errors: None.

See also: arcosh (691), sinh (721), arsinh (693)

Listing: ./mathex/ex8.pp

```
Program Example8;

{ Program to demonstrate the cosh function. }

Uses math;

begin
  Writeln(Cosh(0));
  Writeln(Cosh(1));
end.
```

17.12.15 cot

Synopsis: Alias for `Cotan`

Declaration: `function cot(x: float) : float`

Visibility: default

Description: `cot` is an alias for the cotan (696) function.

See also: cotan (696)

17.12.16 cotan

Synopsis: Return cotangent

Declaration: `function cotan(x: float) : float`

Visibility: default

Description: `Cotan` returns the cotangent of it's argument `x`. `x` should be different from zero.

Errors: If `x` is zero then a overflow error will occur.

See also: tanh (725)

Listing: ./mathex/ex9.pp

```
Program Example9;

{ Program to demonstrate the cotan function. }

Uses math;

begin
  writeln(cotan(pi/2));
  Writeln(cotan(pi/3));
  Writeln(cotan(pi/4));
end.
```

17.12.17 csc

Synopsis: Alias for `cosecant`

Declaration: `function csc(x: float) : float`

Visibility: default

Description: `csc` is an alias for the cosecant (695) function.

See also: cosecant (695)

17.12.18 cycletorad

Synopsis: Convert cycle angle to radians angle

Declaration: `function cycletorad(cycle: float) : float`

Visibility: default

Description: `Cycletorad` transforms it's argument `cycle` (an angle expressed in cycles) to radians. (1 cycle is 2π radians).

Errors: None.

See also: degtograd (697), degtorad (698), radtodeg (716), radtograd (717), radtocycle (716)

Listing: ./mathex/ex10.pp

```
Program Example10;

{ Program to demonstrate the cycletorad function. }

Uses math;

begin
  writeln(cos(cycletorad(1/6))); // Should print 1/2
  writeln(cos(cycletorad(1/8))); // should be sqrt(2)/2
end.
```

17.12.19 degtograd

Synopsis: Convert degree angle to grads angle

Declaration: `function degtograd(deg: float) : float`

Visibility: default

Description: `Degtograd` transforms it's argument `deg` (an angle in degrees) to grads. (90 degrees is 100 grad.)

Errors: None.

See also: cycletorad (697), degtorad (698), radtodeg (716), radtograd (717), radtocycle (716)

Listing: ./mathex/ex11.pp

```
Program Example11;

{ Program to demonstrate the degtograd function. }

Uses math;

begin
  writeln(degtograd(90));
  writeln(degtograd(180));
  writeln(degtograd(270))
end.
```

17.12.20 degtorad

Synopsis: Convert degree angle to radians angle.

Declaration:
```
function degtorad(deg: float) : float
```

Visibility: default

Description: `Degtorad` converts it's argument `deg` (an angle in degrees) to radians. (pi radians is 180 degrees)

Errors: None.

See also: cycletorad (697), degtograd (697), radtodeg (716), radtograd (717), radtocycle (716)

Listing: ./mathex/ex12.pp

```
Program Example12;

{ Program to demonstrate the degtorad function. }

Uses math;

begin
  writeln(degtorad(45));
  writeln(degtorad(90));
  writeln(degtorad(180));
  writeln(degtorad(270));
  writeln(degtorad(360));
end.
```

17.12.21 DivMod

Synopsis: Return DIV and MOD of arguments

Declaration:
```
procedure DivMod(Dividend: Integer;Divisor: Word;var Result: Word;
                 var Remainder: Word)
procedure DivMod(Dividend: Integer;Divisor: Word;var Result: SmallInt;
                 var Remainder: SmallInt)
```

Visibility: default

Description: `DivMod` returns `Dividend` DIV `Divisor` in `Result`, and `Dividend` MOD `Divisor` in `Remainder`

17.12.22 EnsureRange

Synopsis: Change value to it falls in specified range.

Declaration:
```
function EnsureRange(const AValue: Integer;const AMin: Integer;
                     const AMax: Integer) : Integer
function EnsureRange(const AValue: Int64;const AMin: Int64;
                     const AMax: Int64) : Int64
```

Visibility: default

Description: `EnsureRange` returns `Value` if `AValue` is in the range `AMin..AMax`. It returns `AMin` if the value is less than `AMin`, or `AMax` if the value is larger than `AMax`.

See also: InRange (702)

17.12.23 floor

Synopsis: Return the largest integer smaller than or equal to argument

Declaration: `function floor(x: float) : Integer`

Visibility: default

Description: `Floor` returns the largest integer smaller than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If `x` is larger than `maxint`, an overflow will occur.

See also: ceil (694)

Listing: ./mathex/ex13.pp

```
Program Example13;

{ Program to demonstrate the floor function. }

Uses math;

begin
  Writeln(Ceil(-3.7)); // should be -4
  Writeln(Ceil(3.7));  // should be 3
  Writeln(Ceil(-4.0)); // should be -4
end.
```

17.12.24 Frexp

Synopsis: Return mantissa and exponent.

Declaration: `procedure Frexp(X: float;var Mantissa: float;var Exponent: Integer)`

Visibility: default

Description: `Frexp` returns the mantissa and exponent of it's argument `x` in `mantissa` and `exponent`.

Errors: None

Listing: ./mathex/ex14.pp

```
Program Example14;

{ Program to demonstrate the frexp function. }

Uses math;

Procedure dofrexp(Const X : extended);

var man : extended;
    exp: longint;

begin
  man:=0;
  exp:=0;
  frexp(x,man,exp);
  write(x,' has ');
```

```
  WriteIn('mantissa ',man,' and exponent ',exp);
end;


begin
//   dofrexp(1.00);
   dofrexp(1.02e-1);
   dofrexp(1.03e-2);
   dofrexp(1.02e1);
   dofrexp(1.03e2);
end.
```

17.12.25 GetExceptionMask

Synopsis: Get the Floating Point Unit exception mask.

Declaration: `function GetExceptionMask : TFPUExceptionMask`

Visibility: default

Description: Get the Floating Point Unit exception mask.

17.12.26 GetPrecisionMode

Synopsis: Return the Floating Point Unit precision mode.

Declaration: `function GetPrecisionMode : TFPUPrecisionMode`

Visibility: default

Description: Return the Floating Point Unit precision mode.

17.12.27 GetRoundMode

Synopsis: Return the Floating Point Unit rounding mode.

Declaration: `function GetRoundMode : TFPURoundingMode`

Visibility: default

Description: Return the Floating Point Unit rounding mode.

17.12.28 gradtodeg

Synopsis: Convert grads angle to degrees angle

Declaration: `function gradtodeg(grad: float) : float`

Visibility: default

Description: `Gradtodeg` converts its argument `grad` (an angle in grads) to degrees. (100 grad is 90 degrees)

Errors: None.

See also: cycletorad (697), degtograd (697), radtodeg (716), radtograd (717), radtocycle (716), gradtorad (701)

Listing: ./mathex/ex15.pp

```
Program Example15;

{ Program to demonstrate the gradtodeg function. }

Uses math;

begin
  writeln(gradtodeg(100));
  writeln(gradtodeg(200));
  writeln(gradtodeg(300));
end.
```

17.12.29 gradtorad

Synopsis: Convert grads angle to radians angle

Declaration: `function gradtorad(grad: float) : float`

Visibility: default

Description: `Gradtorad` converts its argument `grad` (an angle in grads) to radians. (200 grad is pi degrees).

Errors: None.

See also: cycletorad (697), degtograd (697), radtodeg (716), radtograd (717), radtocycle (716), gradtodeg (700)

Listing: ./mathex/ex16.pp

```
Program Example16;

{ Program to demonstrate the gradtorad function. }

Uses math;

begin
  writeln(gradtorad(100));
  writeln(gradtorad(200));
  writeln(gradtorad(300));
end.
```

17.12.30 hypot

Synopsis: Return hypotenuse of triangle

Declaration: `function hypot(x: float;y: float) : float`

Visibility: default

Description: `Hypot` returns the hypotenuse of the triangle where the sides adjacent to the square angle have lengths `x` and `y`. The function uses Pythagoras' rule for this.

Errors: None.

Listing: ./mathex/ex17.pp

```
Program Example17;

{ Program to demonstrate the hypot function. }

Uses math;

begin
  WriteIn(hypot(3,4)); // should be 5
end.
```

17.12.31 ifthen

Synopsis: Return one of two values, depending on a boolean condition

Declaration:
```
function ifthen(val: Boolean;const iftrue: Integer;
                const iffalse: Integer) : Integer
function ifthen(val: Boolean;const iftrue: Int64;const iffalse: Int64)
                : Int64
function ifthen(val: Boolean;const iftrue: double;const iffalse: double)
                : double
function ifthen(val: Boolean;const iftrue: String;const iffalse: String)
                : String
```

Visibility: default

Description: `ifthen` returns `iftrue` if `val` is `True`, and `False` if `val` is `False`.

This function can be used in expressions.

17.12.32 InRange

Synopsis: Check whether value is in range.

Declaration:
```
function InRange(const AValue: Integer;const AMin: Integer;
                 const AMax: Integer) : Boolean
function InRange(const AValue: Int64;const AMin: Int64;
                 const AMax: Int64) : Boolean
```

Visibility: default

Description: `InRange` returns `True` if `AValue` is in the range `AMin..AMax`. It returns `False` if `Value` lies outside the specified range.

See also: EnsureRange (698)

17.12.33 intpower

Synopsis: Return integer power.

Declaration:
```
function intpower(base: float;const exponent: Integer) : float
```

Visibility: default

Description: `Intpower` returns `base` to the power `exponent`, where exponent is an integer value.

Errors: If `base` is zero and the exponent is negative, then an overflow error will occur.

See also: power (715)

Listing: ./mathex/ex18.pp

```
Program Example18;

{ Program to demonstrate the intpower function. }

Uses math;

Procedure DoIntpower (X : extended; Pow : Integer);

begin
  writeln(X:8:4,'^',Pow:2,' = ',intpower(X,pow):8:4);
end;

begin
  dointpower(0.0,0);
  dointpower(1.0,0);
  dointpower(2.0,5);
  dointpower(4.0,3);
  dointpower(2.0,-1);
  dointpower(2.0,-2);
  dointpower(-2.0,4);
  dointpower(-4.0,3);
end.
```

17.12.34 IsInfinite

Synopsis: Check whether value is infinite

Declaration: `function IsInfinite(const d: Double) : Boolean`

Visibility: default

Description: `IsInfinite` returns `True` if the double `d` contains the infinite value.

See also: IsZero (703), IsInfinite (703)

17.12.35 IsNan

Synopsis: Check whether value is Not a Number

Declaration: `function IsNan(const d: Double) : Boolean`

Visibility: default

Description: `IsNan` returns `True` if the double `d` contains Not A Number (a value which cannot be represented correctly in double format).

See also: IsZero (703), IsInfinite (703)

17.12.36 IsZero

Synopsis: Check whether value is zero

Declaration:
```
function IsZero(const A: Single;Epsilon: Single) : Boolean
function IsZero(const A: Single) : Boolean
function IsZero(const A: Extended;Epsilon: Extended) : Boolean
function IsZero(const A: Extended) : Boolean
```

Visibility: default

Description: `IsZero` checks whether the float value `A` is zero, up to a precision of `Epsilon`. It returns `True` if Abs(`A`) is less than `Epsilon`.

The default value for `Epsilon` is dependent on the type of the arguments, but is MinFloat (687) for the float type.

See also: IsNan (703), IsInfinite (703), SameValue (718)

17.12.37 ldexp

Synopsis: Return (2 to the power p) times x

Declaration:
```
function ldexp(x: float;const p: Integer) : float
```

Visibility: default

Description: `Ldexp` returns (2 to the power `p`) times `x`.

Errors: None.

See also: lnxp1 (704), log10 (705), log2 (705), logn (706)

Listing: ./mathex/ex19.pp

```
Program Example19;

{ Program to demonstrate the ldexp function. }

Uses math;

begin
  writeln(ldexp(2,4):8:4);
  writeln(ldexp(0.5,3):8:4);
end.
```

17.12.38 lnxp1

Synopsis: Return natural logarithm of 1+X

Declaration:
```
function lnxp1(x: float) : float
```

Visibility: default

Description: `Lnxp1` returns the natural logarithm of `1+X`. The result is more precise for small values of `x`. `x` should be larger than -1.

Errors: If $x\leq -1$ then an `EInvalidArgument` exception will be raised.

See also: ldexp (704), log10 (705), log2 (705), logn (706)

Listing: ./mathex/ex20.pp

```
Program Example20;

{ Program to demonstrate the lnxp1 function. }

Uses math;

begin
  writeln(lnxp1(0));
  writeln(lnxp1(0.5));
  writeln(lnxp1(1));
end.
```

17.12.39 log10

Synopsis: Return 10-Based logarithm.

Declaration: `function log10(x: float) : float`

Visibility: default

Description: `Log10` returns the 10-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (704), lnxp1 (704), log2 (705), logn (706)

Listing: ./mathex/ex21.pp

```
Program Example21;

{ Program to demonstrate the log10 function. }

Uses math;

begin
  Writeln(Log10(10):8:4);
  Writeln(Log10(100):8:4);
  Writeln(Log10(1000):8:4);
  Writeln(Log10(1):8:4);
  Writeln(Log10(0.1):8:4);
  Writeln(Log10(0.01):8:4);
  Writeln(Log10(0.001):8:4);
end.
```

17.12.40 log2

Synopsis: Return 2-based logarithm

Declaration: `function log2(x: float) : float`

Visibility: default

Description: `Log2` returns the 2-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (704), lnxp1 (704), log10 (705), logn (706)

Listing: ./mathex/ex22.pp

```
Program Example22;

{ Program to demonstrate the log2 function. }

Uses math;

begin
  Writeln(Log2(2):8:4);
  Writeln(Log2(4):8:4);
  Writeln(Log2(8):8:4);
  Writeln(Log2(1):8:4);
  Writeln(Log2(0.5):8:4);
  Writeln(Log2(0.25):8:4);
  Writeln(Log2(0.125):8:4);
end.
```

17.12.41 logn

Synopsis: Return N-based logarithm.

Declaration:
```
function logn(n: float;x: float) : float
```

Visibility: default

Description: `Logn` returns the n-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (704), lnxp1 (704), log10 (705), log2 (705)

Listing: ./mathex/ex23.pp

```
Program Example23;

{ Program to demonstrate the logn function. }

Uses math;

begin
  Writeln(Logn(3,4):8:4);
  Writeln(Logn(2,4):8:4);
  Writeln(Logn(6,9):8:4);
  Writeln(Logn(exp(1),exp(1)):8:4);
  Writeln(Logn(0.5,1):8:4);
  Writeln(Logn(0.25,3):8:4);
  Writeln(Logn(0.125,5):8:4);
end.
```

17.12.42 Max

Synopsis: Return largest of 2 values

Declaration:
```
function Max(a: Integer;b: Integer) : Integer
function Max(a: Int64;b: Int64) : Int64
function Max(a: Extended;b: Extended) : Extended
```

Visibility: default

Description: `Max` returns the maximum of `Int1` and `Int2`.

Errors: None.

See also: min (710), maxIntValue (707), maxvalue (708)

Listing: ./mathex/ex24.pp

```
Program Example24;

{ Program to demonstrate the max function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln(max(a,b));
end.
```

17.12.43 MaxIntValue

Synopsis: Return largest element in integer array

Declaration: `function MaxIntValue(const Data: Array of Integer) : Integer`

Visibility: default

Description: `MaxIntValue` returns the largest integer out of the `Data` array.

This function is provided for Delphi compatibility, use the maxvalue (708) function instead.

Errors: None.

See also: maxvalue (708), minvalue (711), minIntValue (711)

Listing: ./mathex/ex25.pp

```
Program Example25;

{ Program to demonstrate the MaxIntValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}

Uses math;

Type
  TExArray = Array[1..100] of Integer;

Var
  I : Integer;
  ExArray : TExArray;

begin
```

```
  Randomize;
  for I:=low(exarray) to high(exarray) do
    ExArray[i]:=Random(I)-Random(100);
  WriteIn(MaxIntValue(ExArray));
end.
```

17.12.44 maxvalue

Synopsis: Return largest value in array

Declaration:
```
function maxvalue(const data: Array of Extended) : Extended
function maxvalue(const data: PExtended;const N: Integer) : Extended
function maxvalue(const data: Array of Integer) : Integer
function maxvalue(const data: PInteger;const N: Integer) : Integer
```

Visibility: default

Description: `Maxvalue` returns the largest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: maxIntValue (707), minvalue (711), minIntValue (711)

Listing: ./mathex/ex26.pp

```
program Example26;

{ Program to demonstrate the MaxValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}

uses math;

var i:1..100;
    f_array:array[1..100] of Float;
    i_array:array[1..100] of Integer;
    Pf_array:Pfloat;
    PI_array:Pinteger;

begin
  randomize;

  Pf_array:=@f_array[1];
  Pi_array:=@i_array[1];

  for i:=low(f_array) to high(f_array) do
    f_array[i]:=(random-random)*100;
  for i:=low(i_array) to high(i_array) do
    i_array[i]:=random(I)-random(100);

  WriteIn('Max Float       : ',MaxValue(f_array):8:4);
  WriteIn('Max Float   (b) : ',MaxValue(Pf_array,100):8:4);
  WriteIn('Max Integer     : ',MaxValue(i_array):8);
  WriteIn('Max Integer (b) : ',MaxValue(Pi_array,100):8);
end.
```

17.12.45 mean

Synopsis: Return mean value of array

Declaration:
```
function mean(const data: Array of Extended) : float
function mean(const data: PExtended;const N: LongInt) : float
```

Visibility: default

Description: `Mean` returns the average value of `data`. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: meanandstddev (709), momentskewkurtosis (712), sum (722)

Listing: ./mathex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Mean function. }

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max       : ',MaxValue(ExArray):8:4);
  Writeln('Min       : ',MinValue(ExArray):8:4);
  Writeln('Mean      : ',Mean(ExArray):8:4);
  Writeln('Mean (b)  : ',Mean(@ExArray[1],100):8:4);
end.
```

17.12.46 meanandstddev

Synopsis: Return mean and standard deviation of array

Declaration:
```
procedure meanandstddev(const data: Array of Extended;var mean: float;
                        var stddev: float)
procedure meanandstddev(const data: PExtended;const N: LongInt;
                        var mean: float;var stddev: float)
```

Visibility: default

Description: `meanandstddev` calculates the mean and standard deviation of `data` and returns the result in `mean` and `stddev`, respectively. Stddev is zero if there is only one value. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: mean (709), sum (722), sumofsquares (723), momentskewkurtosis (712)

Listing: ./mathex/ex28.pp

```
Program Example28;

{ Program to demonstrate the Meanandstddev function. }

Uses math;

Type
  TExArray = Array[1..100] of Extended;

Var
  I : Integer;
  ExArray : TExArray;
  Mean,stddev : Extended;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  MeanAndStdDev(ExArray,Mean,StdDev);
  Writeln('Mean      : ',Mean:8:4);
  Writeln('StdDev    : ',StdDev:8:4);
  MeanAndStdDev(@ExArray[1],100,Mean,StdDev);
  Writeln('Mean (b)  : ',Mean:8:4);
  Writeln('StdDev (b) : ',StdDev:8:4);
end.
```

17.12.47 Min

Synopsis: Return smallest of two values.

Declaration:
```
function Min(a: Integer;b: Integer) : Integer
function Min(a: Int64;b: Int64) : Int64
function Min(a: Extended;b: Extended) : Extended
```

Visibility: default

Description: `min` returns the smallest value of `Int1` and `Int2`;

Errors: None.

See also: max (706)

Listing: ./mathex/ex29.pp

```
Program Example29;

{ Program to demonstrate the min function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln(min(a,b));
end.
```

17.12.48 MinIntValue

Synopsis: Return smallest value in integer array

Declaration:
```
function MinIntValue(const Data: Array of Integer) : Integer
```

Visibility: default

Description: `MinIntvalue` returns the smallest value in the `Data` array.

This function is provided for Delphi compatibility, use `minvalue` instead.

Errors: None

See also: minvalue (711), maxIntValue (707), maxvalue (708)

Listing: ./mathex/ex30.pp

```
Program Example30;

{ Program to demonstrate the MinIntValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}

Uses math;

Type
  TExArray = Array[1..100] of Integer;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Random(I)-Random(100);
  Writeln(MinIntValue(ExArray));
end.
```

17.12.49 minvalue

Synopsis: Return smallest value in array

Declaration:
```
function minvalue(const data: Array of Extended) : Extended
function minvalue(const data: PExtended;const N: Integer) : Extended
function minvalue(const data: Array of Integer) : Integer
function MinValue(const Data: PInteger;const N: Integer) : Integer
```

Visibility: default

Description: `Minvalue` returns the smallest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: maxIntValue (707), maxvalue (708), minIntValue (711)

Listing: ./mathex/ex31.pp

```
program Example31;

{ Program to demonstrate the MinValue function. }

{ Make sure integer is 32 bit}
{$mode objfpc}

uses math;

var i:1..100;
    f_array:array[1..100] of Float;
    i_array:array[1..100] of Integer;
    Pf_array:Pfloat;
    PI_array:Pinteger;

begin
  randomize;

  Pf_array:=@f_array[1];
  Pi_array:=@i_array[1];

  for i:=low(f_array) to high(f_array) do
    f_array[i]:=(random-random)*100;
  for i:=low(i_array) to high(i_array) do
    i_array[i]:=random(l)-random(100);

  Writeln('Min Float       : ',MinValue(f_array):8:4);
  Writeln('Min Float   (b) : ',MinValue(Pf_array,100):8:4);
  Writeln('Min Integer     : ',MinValue(i_array):8);
  Writeln('Min Integer (b) : ',MinValue(Pi_array,100):8);
end.
```

17.12.50 momentskewkurtosis

Synopsis: Return 4 first moments of distribution

Declaration:
```
procedure momentskewkurtosis(const data: Array of Extended;
                             out m1: float;out m2: float;out m3: float;
                             out m4: float;out skew: float;
                             out kurtosis: float)
procedure momentskewkurtosis(const data: PExtended;const N: Integer;
                             out m1: float;out m2: float;out m3: float;
                             out m4: float;out skew: float;
                             out kurtosis: float)
```

Visibility: default

Description: `momentskewkurtosis` calculates the 4 first moments of the distribution of valuesin `data` and returns them in `m1`,`m2`,`m3` and `m4`, as well as the `skew` and `kurtosis`.

Errors: None.

See also: mean (709), meanandstddev (709)

Listing: ./mathex/ex32.pp

```
program Example32;

{ Program to demonstrate the momentskewkurtosis function. }

uses math;

var distarray:array[1..1000] of float;
    I:longint;
    m1,m2,m3,m4,skew,kurtosis:float;

begin
  randomize;
  for I:=low(distarray) to high(distarray) do
    distarray[i]:=random;
  momentskewkurtosis(DistArray,m1,m2,m3,m4,skew,kurtosis);

  Writeln('1st moment : ',m1:8:6);
  Writeln('2nd moment : ',m2:8:6);
  Writeln('3rd moment : ',m3:8:6);
  Writeln('4th moment : ',m4:8:6);
  Writeln('Skew       : ',skew:8:6);
  Writeln('kurtosis   : ',kurtosis:8:6);
end.
```

17.12.51 norm

Synopsis: Return Euclidian norm

Declaration:
```
function norm(const data: Array of Extended) : float
function norm(const data: PExtended;const N: Integer) : float
```

Visibility: default

Description: `Norm` calculates the Euclidian norm of the array of data. This equals `sqrt(sumofsquares(data))`.

The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: sumofsquares (723)

Listing: ./mathex/ex33.pp

```
program Example33;

{ Program to demonstrate the norm function. }

uses math;

var v:array[1..10] of Float;
    I:1..10;

begin
  for I:=low(v) to high(v) do
    v[i]:=random;
  writeln(norm(v));
end.
```

17.12.52 operator **(float, float): float

Declaration: `function operator **(float, float): float(bas: float;expo: float) : float`

Visibility: default

17.12.53 operator **(Int64, Int64): Int64

Declaration: `function operator **(Int64, Int64): Int64(bas: Int64;expo: Int64) : Int64`

Visibility: default

17.12.54 popnstddev

Synopsis: Return population variance

Declaration:
```
function popnstddev(const data: Array of Extended) : float
function popnstddev(const data: PExtended;const N: Integer) : float
```

Visibility: default

Description: `Popnstddev` returns the square root of the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: popnvariance (715), mean (709), meanandstddev (709), stddev (722), momentskewkurtosis (712)

Listing: ./mathex/ex35.pp

```
Program Example35;

{ Program to demonstrate the PopnStdDev function. }

Uses Math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max               : ',MaxValue(ExArray):8:4);
  Writeln('Min               : ',MinValue(ExArray):8:4);
  Writeln('Pop. stddev.      : ',PopnStdDev(ExArray):8:4);
  Writeln('Pop. stddev. (b) : ',PopnStdDev(@ExArray[1],100):8:4);
end.
```

17.12.55 popnvariance

Synopsis: Return population variance

Declaration:
```
function popnvariance(const data: PExtended;const N: Integer) : float
function popnvariance(const data: Array of Extended) : float
```

Visibility: default

Description: `Popnvariance` returns the square root of the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: popnstddev (714), mean (709), meanandstddev (709), stddev (722), momentskewkurtosis (712)

Listing: ./mathex/ex36.pp

```
Program Example36;

{ Program to demonstrate the PopnVariance function. }

Uses math;

Var
  I : Integer;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max            : ',MaxValue(ExArray):8:4);
  Writeln('Min            : ',MinValue(ExArray):8:4);
  Writeln('Pop. var.      : ',PopnVariance(ExArray):8:4);
  Writeln('Pop. var. (b) : ',PopnVariance(@ExArray[1],100):8:4);
end.
```

17.12.56 power

Synopsis: Return real power.

Declaration:
```
function power(base: float;exponent: float) : float
```

Visibility: default

Description: `power` raises `base` to the power `power`. This is equivalent to `exp(power*ln(base))`. Therefore `base` should be non-negative.

Errors: None.

See also: intpower (702)

Listing: ./mathex/ex34.pp

```
Program Example34;

{ Program to demonstrate the power function. }

Uses Math;

procedure dopower(x,y : float);

begin
  writeln(x:8:6,'^',y:8:6,' = ',power(x,y):8:6)
end;

begin
  dopower(2,2);
  dopower(2,-2);
  dopower(2,0.0);
end.
```

17.12.57 radtocycle

Synopsis: Convert radians angle to cycle angle

Declaration: `function radtocycle(rad: float) : float`

Visibility: default

Description: `Radtocycle` converts its argument `rad` (an angle expressed in radians) to an angle in cycles. (1 cycle equals 2 `pi` radians)

Errors: None.

See also: degtograd (697), degtorad (698), radtodeg (716), radtograd (717), cycletorad (697)

Listing: ./mathex/ex37.pp

```
Program Example37;

{ Program to demonstrate the radtocycle function. }

Uses math;

begin
  writeln(radtocycle(2*pi):8:6);
  writeln(radtocycle(pi):8:6);
  writeln(radtocycle(pi/2):8:6);
end.
```

17.12.58 radtodeg

Synopsis: Convert radians angle to degrees angle

Declaration: `function radtodeg(rad: float) : float`

Visibility: default

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in degrees. (180 degrees equals pi radians)

Errors: None.

See also: degtograd (697), degtorad (698), radtocycle (716), radtograd (717), cycletorad (697)

Listing: ./mathex/ex38.pp

```
Program Example38;

{ Program to demonstrate the radtodeg function. }

Uses math;

begin
  writeln(radtodeg(2*pi):8:6);
  writeln(radtodeg(pi):8:6);
  writeln(radtodeg(pi/2):8:6);
end.
```

17.12.59 radtograd

Synopsis: Convert radians angle to grads angle

Declaration: `function radtograd(rad: float) : float`

Visibility: default

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in grads. (200 grads equals pi radians)

Errors: None.

See also: degtograd (697), degtorad (698), radtocycle (716), radtodeg (716), cycletorad (697)

Listing: ./mathex/ex39.pp

```
Program Example39;

{ Program to demonstrate the radtograd function. }

Uses math;

begin
  writeln(radtograd(2*pi):8:6);
  writeln(radtograd(pi):8:6);
  writeln(radtograd(pi/2):8:6);
end.
```

17.12.60 randg

Synopsis: Return gaussian distributed random number.

Declaration: `function randg(mean: float;stddev: float) : float`

Visibility: default

Description: `randg` returns a random number which - when produced in large quantities - has a Gaussian distribution with mean `mean` and standarddeviation `stddev`.

Errors: None.

See also: mean (709), stddev (722), meanandstddev (709)

Listing: ./mathex/ex40.pp

```
Program Example40;

{ Program to demonstrate the randg function. }

Uses Math;

Var
  I : Integer;
  ExArray : Array[1..10000] of Float;;
  Mean,stddev : Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Randg(1,0.2);
  MeanAndStdDev(ExArray,Mean,StdDev);
  Writeln('Mean      : ',Mean:8:4);
  Writeln('StdDev    : ',StdDev:8:4);
end.
```

17.12.61 RoundTo

Synopsis: Round to the specified number of digits

Declaration:
```
function RoundTo(const AVAlue: Extended;const Digits: TRoundToRange)
                 : Extended
```

Visibility: default

Description: `RoundTo` rounds the specified float `AVAlue` to the specified number of digits and returns the result. This result is accurate to "10 to the power Digits". It uses the standard `Round` function for this.

See also: TRoundToRange (689), SimpleRoundTo (720)

17.12.62 SameValue

Synopsis: Check whether 2 float values are the same

Declaration:
```
function SameValue(const A: Extended;const B: Extended) : Boolean
function SameValue(const A: Single;const B: Single) : Boolean
function SameValue(const A: Extended;const B: Extended;
                   Epsilon: Extended) : Boolean
function SameValue(const A: Single;const B: Single;Epsilon: Single)
                   : Boolean
```

Visibility: default

Description: `SameValue` returns `True` if the floating-point values `A` and `B` are the same, i.e. whether the absolute value of their their difference is smaller than `Epsilon`. If their difference is larger, then `False` is returned.

The default value for `Epsilon` is dependent on the type of the arguments, but is MinFloat (687) for the float type.

See also: MinFloat (687), IsZero (703)

17.12.63 sec

Synopsis: Alias for `secant`

Declaration:
```
function sec(x: float) : float
```

Visibility: default

Description: `sec` is an alias for the secant (719) function.

See also: secant (719)

17.12.64 secant

Synopsis: Calculate secant

Declaration:
```
function secant(x: float) : float
```

Visibility: default

Description: `Secant` calculates the secant (1/`cos(x)`) of its argument `x`.

Errors: If 90 or 270 degrees is specified, an exception will be raised.

See also: cosecant (695)

17.12.65 SetExceptionMask

Synopsis: Set the Floating Point Unit exception mask.

Declaration:
```
function SetExceptionMask(const Mask: TFPUExceptionMask)
                                    : TFPUExceptionMask
```

Visibility: default

Description: Set the Floating Point Unit exception mask.

17.12.66 SetPrecisionMode

Synopsis: Set the Floating Point Unit precision mode.

Declaration:
```
function SetPrecisionMode(const Precision: TFPUPrecisionMode)
                                    : TFPUPrecisionMode
```

Visibility: default

Description: Set the Floating Point Unit precision mode.

17.12.67 SetRoundMode

Synopsis: Set the Floating Point Unit rounding mode.

Declaration:
```
function SetRoundMode(const RoundMode: TFPURoundingMode)
                            : TFPURoundingMode
```

Visibility: default

Description: Set the Floating Point Unit rounding mode.

17.12.68 Sign

Synopsis: Return sign of argument

Declaration:
```
function Sign(const AValue: Integer) : TValueSign
function Sign(const AValue: Int64) : TValueSign
function Sign(const AValue: Double) : TValueSign
function Sign(const AValue: Extended) : TValueSign
```

Visibility: default

Description: `Sign` returns the sign of it's argument, which can be an Integer, 64 bit integer, or a double. The returned value is an integer which is -1, 0 or 1, and can be used to do further calculations with.

17.12.69 SimpleRoundTo

Synopsis: Round to the specified number of digits (rounding up if needed)

Declaration:
```
function SimpleRoundTo(const AValue: Extended;
                       const Digits: TRoundToRange) : Extended
```

Visibility: default

Description: `SimplRoundTo` rounds the specified float `AVAlue` to the specified number of digits, but rounds up, and returns the result. This result is accurate to "10 to the power Digits". It uses the standard `Round` function for this.

See also: TRoundToRange (689), RoundTo (718)

17.12.70 sincos

Synopsis: Return sine and cosine of argument

Declaration:
```
procedure sincos(theta: float;out sinus: float;out cosinus: float)
```

Visibility: default

Description: `Sincos` calculates the sine and cosine of the angle `theta`, and returns the result in `sinus` and `cosinus`.

On Intel hardware, This calculation will be faster than making 2 calls to calculate the sine and cosine separately.

Errors: None.

See also: arcsin (691), arccos (690)

Listing: ./mathex/ex41.pp

```
Program Example41;

{ Program to demonstrate the sincos function. }

Uses math;

Procedure dosincos(Angle : Float);

Var
  Sine,Cosine : Float;

begin
  sincos(angle,sine,cosine);
  Write('Angle : ',Angle:8:6);
  Write(' Sine :',sine:8:6);
  Write(' Cosine :',cosine:8:6);
end;

begin
  dosincos(pi);
  dosincos(pi/2);
  dosincos(pi/3);
  dosincos(pi/4);
  dosincos(pi/6);
end.
```

17.12.71 sinh

Synopsis: Return hyperbolic sine

Declaration: `function sinh(x: float) : float`

Visibility: default

Description: `Sinh` returns the hyperbolic sine of its argument `x`.

Errors:

See also: cosh (695), arsinh (693), tanh (725), artanh (693)

Listing: ./mathex/ex42.pp

```
Program Example42;

{ Program to demonstrate the sinh function. }

Uses math;

begin
  writeln(sinh(0));
  writeln(sinh(1));
  writeln(sinh(-1));
end.
```

17.12.72 stddev

Synopsis: Return standard deviation of data

Declaration:
```
function stddev(const data: Array of Extended) : float
function stddev(const data: PExtended;const N: Integer) : float
```

Visibility: default

Description: `Stddev` returns the standard deviation of the values in `Data`. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: mean (709), meanandstddev (709), variance (727), totalvariance (726)

Listing: ./mathex/ex43.pp

```
Program Example40;

{ Program to demonstrate the stddev function. }

Uses Math;

Var
  I : Integer;
  ExArray : Array[1..10000] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Randg(1,0.2);
  Writeln('StdDev      : ',StdDev(ExArray):8:4);
  Writeln('StdDev (b) : ',StdDev(@ExArray[0],10000):8:4);
end.
```

17.12.73 sum

Synopsis: Return sum of values

Declaration:
```
function sum(const data: Array of Extended) : float
function sum(const data: PExtended;const N: LongInt) : float
```

Visibility: default

Description: `Sum` returns the sum of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sumofsquares (723), sumsandsquares (724), totalvariance (726), variance (727)

Listing: ./mathex/ex44.pp

```
Program Example44;

{ Program to demonstrate the Sum function. }

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max      : ',MaxValue(ExArray):8:4);
  Writeln('Min      : ',MinValue(ExArray):8:4);
  Writeln('Sum      : ',Sum(ExArray):8:4);
  Writeln('Sum (b) : ',Sum(@ExArray[1],100):8:4);
end.
```

17.12.74 sumInt

Synopsis: Return the sum of an array of integers

Declaration:
```
function sumInt(const data: PInt64;const N: LongInt) : Int64
function sumInt(const data: Array of Int64) : Int64
```

Visibility: default

Description: `SumInt` returns the sum of the `N` integers in the `Data` array, where this can be an open array or a pointer to an array.

Errors: An overflow may occur.

17.12.75 sumofsquares

Synopsis: Return sum of squares of values

Declaration:
```
function sumofsquares(const data: Array of Extended) : float
function sumofsquares(const data: PExtended;const N: Integer) : float
```

Visibility: default

Description: `Sumofsquares` returns the sum of the squares of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sum (722), sumsandsquares (724), totalvariance (726), variance (727)

Listing: ./mathex/ex45.pp

```
Program Example45;

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

```
Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max              : ',MaxValue(ExArray):8:4);
  Writeln('Min              : ',MinValue(ExArray):8:4);
  Writeln('Sum squares      : ',SumOfSquares(ExArray):8:4);
  Writeln('Sum squares (b) : ',SumOfSquares(@ExArray[1],100):8:4);
end.
```

17.12.76 sumsandsquares

Synopsis: Return sum and sum of squares of values.

Declaration:
```
procedure sumsandsquares(const data: Array of Extended;var sum: float;
                         var sumofsquares: float)
procedure sumsandsquares(const data: PExtended;const N: Integer;
                         var sum: float;var sumofsquares: float)
```

Visibility: default

Description: `sumsandsquares` calculates the sum of the values and the sum of the squares of the values in the `data` array and returns the results in `sum` and `sumofsquares`.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sum (722), sumofsquares (723), totalvariance (726), variance (727)

Listing: ./mathex/ex46.pp

```
Program Example45;

{ Program to demonstrate the SumOfSquares function. }

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float
  s,ss : float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max              : ',MaxValue(ExArray):8:4);
  Writeln('Min              : ',MinValue(ExArray):8:4);
  SumsAndSquares(ExArray,S,SS);
  Writeln('Sum              : ',S:8:4);
  Writeln('Sum squares      : ',SS:8:4);
```

```
  SumsAndSquares(@ExArray[1],100,S,SS);
  Writeln('Sum (b)          : ',S:8:4);
  Writeln('Sum squares (b) : ',SS:8:4);
end.
```

17.12.77 tan

Synopsis: Return tangent

Declaration: `function tan(x: float) : float`

Visibility: default

Description: `Tan` returns the tangent of `x`.

Errors: If `x` (normalized) is pi/2 or 3pi/2 then an overflow will occur.

See also: tanh (725), arcsin (691), sincos (720), arccos (690)

Listing: ./mathex/ex47.pp

```
Program Example47;

{ Program to demonstrate the Tan function. }

Uses math;

Procedure DoTan(Angle : Float);

begin
  Write('Angle : ',RadToDeg(Angle):8:6);
  Writeln(' Tangent : ',Tan(Angle):8:6);
end;

begin
  DoTan(0);
  DoTan(Pi);
  DoTan(Pi/3);
  DoTAn(Pi/4);
  DoTan(Pi/6);
end.
```

17.12.78 tanh

Synopsis: Return hyperbolic tangent

Declaration: `function tanh(x: float) : float`

Visibility: default

Description: `Tanh` returns the hyperbolic tangent of `x`.

Errors: None.

See also: arcsin (691), sincos (720), arccos (690)

Listing: ./mathex/ex48.pp

```
Program Example48;

{ Program to demonstrate the Tanh function. }

Uses math;

begin
  writeln(tanh(0));
  writeln(tanh(1));
  writeln(tanh(-1));
end.
```

17.12.79 totalvariance

Synopsis: Return total varians of values

Declaration:
```
function totalvariance(const data: Array of Extended) : float
function totalvariance(const data: PExtended;const N: Integer) : float
```

Visibility: default

Description: `TotalVariance` returns the total variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: variance (727), stddev (722), mean (709)

Listing: ./mathex/ex49.pp

```
Program Example49;

{ Program to demonstrate the TotalVariance function. }

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;
  TV : float;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[i]:=(Random-Random)*100;
  TV:=TotalVariance(ExArray);
  Writeln('Total variance      : ',TV:8:4);
  TV:=TotalVariance(@ExArray[1],100);
  Writeln('Total Variance (b) : ',TV:8:4);
end.
```

17.12.80 variance

Synopsis: Return variance of values

Declaration:
```
function variance(const data: Array of Extended) : float
function variance(const data: PExtended;const N: Integer) : float
```

Visibility: default

Description: `Variance` returns the variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: totalvariance (726), stddev (722), mean (709)

Listing: ./mathex/ex50.pp

```
Program Example50;

{ Program to demonstrate the Variance function. }

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;
  V : float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  V:=Variance(ExArray);
  Writeln('Variance      : ',V:8:4);
  V:=Variance(@ExArray[1],100);
  Writeln('Variance (b) : ',V:8:4);
end.
```

17.13 einvalidargument

17.13.1 Description

Exception raised when invalid arguments are passed to a function.

Chapter 18

Reference for unit 'matrix'

18.1 Overview

The unit matrix is a unit that provides objects for the common two, three and four dimensional vectors matrixes. These vectors and matrixes are very common in computer graphics and are often implemented from scratch by programmers while every implementation provides exactly the same functionality.

It makes therefore sense to provide this functionality in the runtime library. This eliminates the need for programmers to reinvent the wheel and also allows libraries that use matrix operations to become more compatible.

The matrix unit does not provide n-dimensional matrixes. The functionality needs of a general matrix unit varies from application to application; one can think of reduced memory usage tricks for matrixes that only have data around the diagonal etc., desire for parallelization etc. etc. It is believed that programmers that do use n-dimensional matrices would not necessarily benfit from such a unit in the runtime library.

Design goals:

- Provide common dimensions, two three and four.
- Provide multiple floating point precisions, single, double, extended.
- Simple trivial binary representation; it is possible to typecast vectors into other implementations that use the same trivial representation.
- No dynamic memory management in the background. It must be possible to write expressions like matrix A * B * C without worrying about memory management.

Design decisions:

- Class object model is ruled out. The objects object model, without virtual methods, is suitable.
- Operator overloading is a good way to allow programmers to write matrix expressions.
- 3 dimensions * 3 precision means 9 vector and 9 matrix objects. Macro's have been used in the source to take care of this.

18.2 Constants, types and variables

18.2.1 Types

```
Tmatrix2_double_data = Array[0..1,0..1] of double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_extended_data = Array[0..1,0..1] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_single_data = Array[0..1,0..1] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_double_data = Array[0..2,0..2] of double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_extended_data = Array[0..2,0..2] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_single_data = Array[0..2,0..2] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_double_data = Array[0..3,0..3] of double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_extended_data = Array[0..3,0..3] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_single_data = Array[0..3,0..3] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_double_data = Array[0..1] of double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_extended_data = Array[0..1] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_single_data = Array[0..1] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_double_data = Array[0..2] of double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_extended_data = Array[0..2] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_single_data = Array[0..2] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_double_data = Array[0..3] of double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_extended_data = Array[0..3] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_single_data = Array[0..3] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

18.3 Procedures and functions

18.3.1 operator *(Tmatrix2_double, double): Tmatrix2_double

Synopsis: Multiply a two-dimensional double precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix2_double, double): Tmatrix2_double
                                                (const m: Tmatrix2_doub
                                               const x: double)
                                                : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.2 operator *(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double

Synopsis: Give product of two two-dimensional double precision matrices

Declaration:
```
function operator *(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double
                                                        (const m1: Tma
                                                       const m2: Tmat
                                                        : Tmatrix2_do
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.3 operator *(Tmatrix2_double, Tvector2_double): Tvector2_double

Synopsis: Give product of a two-dimendional double precision matrix and vector

Declaration:
```
function operator *(Tmatrix2_double, Tvector2_double): Tvector2_double
                                                        (const m: Tmat
                                                        const v: Tvect
                                                         : Tvector2_do
```

Visibility: default

Description: This operator allows you to multipy a two-dimensional double precision matrices with a two dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.4 operator *(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Multiply a two-dimensional extended precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix2_extended, extended): Tmatrix2_extended
                                                   (const m: Tmatrix
                                                   const x: extended
                                                    : Tmatrix2_exten
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.5 operator *(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended

Synopsis: Give product of two two-dimensional extended precision matrices

Declaration:
```
                                                   function
                                                   (const m
                                                   const m2
                                                    : Tmatr
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.6 operator *(Tmatrix2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Give product of a two-dimendional extended precision matrix and vector

Declaration:
```
                                                   function
                                                   (const m
                                                   const v:
                                                    : Tvect
```

Visibility: default

Description: This operator allows you to multipy a two-dimensional extended precision matrices with a two dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.7 operator *(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Multiply a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix2_single, single): Tmatrix2_single
                                                    (const m: Tmatrix2_sing
                                                    const x: single)
                                                     : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.8 operator *(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single

Synopsis: Give product of two two-dimensional single precision matrices

Declaration:
```
function operator *(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single
                                                          (const m1: Tma
                                                          const m2: Tmat
                                                           : Tmatrix2_si
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.9 operator *(Tmatrix2_single, Tvector2_single): Tvector2_single

Synopsis: Give product of a two-dimendional single precision matrix and vector

Declaration:
```
function operator *(Tmatrix2_single, Tvector2_single): Tvector2_single
                                                          (const m: Tmat
                                                          const v: Tvect
                                                           : Tvector2_si
```

Visibility: default

Description: This operator allows you to multipy a two-dimensional single precision matrices with a two dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.10 operator *(Tmatrix3_double, double): Tmatrix3_double

Synopsis: Multiply a three-dimensional double precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix3_double, double): Tmatrix3_double
                                                    (const m: Tmatrix3_doub
                                                    const x: double)
                                                     : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.11 operator *(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double

Synopsis: Give product of two three-dimensional double precision matrices

Declaration:
```
function operator *(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double
                                                                (const m1: Tma
                                                               const m2: Tmat
                                                                : Tmatrix3_do
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.12 operator *(Tmatrix3_double, Tvector3_double): Tvector3_double

Synopsis: Give product of a three-dimendional double precision matrix and vector

Declaration:
```
function operator *(Tmatrix3_double, Tvector3_double): Tvector3_double
                                                                (const m: Tmat
                                                               const v: Tvect
                                                                : Tvector3_do
```

Visibility: default

Description: This operator allows you to multipy a three-dimensional double precision matrices with a three dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.13 operator *(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Multiply a three-dimensional extended precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix3_extended, extended): Tmatrix3_extended
                                                             (const m: Tmatrix
                                                            const x: extended
                                                             : Tmatrix3_exten
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.14 operator *(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended

Synopsis: Give product of two three-dimensional extended precision matrices

Declaration:

```
function
(const m
const m2
 : Tmatr
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.15 operator *(Tmatrix3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Give product of a three-dimendional extended precision matrix and vector

Declaration:

```
function
(const m
const v:
 : Tvect
```

Visibility: default

Description: This operator allows you to multipy a three-dimensional extended precision matrices with a three dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.16 operator *(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Multiply a three-dimensional single precision matrix bye a scalar

Declaration:

```
function operator *(Tmatrix3_single, single): Tmatrix3_single
                                              (const m: Tmatrix3_sing
                                              const x: single)
                                               : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.17 operator *(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single

Synopsis: Give product of two three-dimensional single precision matrices

Declaration:

```
function operator *(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single
                                                       (const m1: Tma
                                                       const m2: Tmat
                                                        : Tmatrix3_si
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.18 operator *(Tmatrix3_single, Tvector3_single): Tvector3_single

Synopsis: Give product of a three-dimendional single precision matrix and vector

Declaration:
```
function operator *(Tmatrix3_single, Tvector3_single): Tvector3_single
                                                        (const m: Tmat
                                                        const v: Tvect
                                                         : Tvector3_si
```

Visibility: default

Description: This operator allows you to multipy a three-dimensional single precision matrices with a three dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.19 operator *(Tmatrix4_double, double): Tmatrix4_double

Synopsis: Multiply a four-dimensional double precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix4_double, double): Tmatrix4_double
                                               (const m: Tmatrix4_doub
                                               const x: double)
                                                : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.20 operator *(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double

Synopsis: Give product of two four-dimensional double precision matrices

Declaration:
```
function operator *(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double
                                                        (const m1: Tma
                                                        const m2: Tmat
                                                         : Tmatrix4_do
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.21 operator *(Tmatrix4_double, Tvector4_double): Tvector4_double

Synopsis: Give product of a four-dimendional double precision matrix and vector

Declaration:
```
function operator *(Tmatrix4_double, Tvector4_double): Tvector4_double
                                                        (const m: Tmat
                                                        const v: Tvect
                                                         : Tvector4_do
```

Visibility: default

Description: This operator allows you to multipy a four-dimensional double precision matrices with a four dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.22 operator *(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Multiply a four-dimensional extended precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix4_extended, extended): Tmatrix4_extended
                                                  (const m: Tmatrix
                                                  const x: extended
                                                   : Tmatrix4_exten
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.23 operator *(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended

Synopsis: Give product of two four-dimensional extended precision matrices

Declaration:
```
function
(const m
const m2
 : Tmatr
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.24 operator *(Tmatrix4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Give product of a four-dimendional extended precision matrix and vector

Declaration:
```
function
(const m
const v:
 : Tvect
```

Visibility: default

Description: This operator allows you to multipy a four-dimensional extended precision matrices with a four dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.25 operator *(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Multiply a four-dimensional single precision matrix bye a scalar

Declaration:
```
function operator *(Tmatrix4_single, single): Tmatrix4_single
                                              (const m: Tmatrix4_sing
                                              const x: single)
                                               : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

18.3.26 operator *(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single

Synopsis: Give product of two four-dimensional single precision matrices

Declaration:
```
function operator *(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single
                                                        (const m1: Tma
                                                        const m2: Tmat
                                                         : Tmatrix4_si
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

18.3.27 operator *(Tmatrix4_single, Tvector4_single): Tvector4_single

Synopsis: Give product of a four-dimendional single precision matrix and vector

Declaration:
```
function operator *(Tmatrix4_single, Tvector4_single): Tvector4_single
                                                        (const m: Tmat
                                                        const v: Tvect
                                                         : Tvector4_si
```

Visibility: default

Description: This operator allows you to multipy a four-dimensional single precision matrices with a four dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

18.3.28 operator *(Tvector2_double, double): Tvector2_double

Synopsis: Multiply a two-dimensional double precision vector by a scalar

Declaration:
```
function operator *(Tvector2_double, double): Tvector2_double
                                               (const x: Tvector2_doub
                                               y: double)
                                                : Tvector2_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.29 operator *(Tvector2_double, Tvector2_double): Tvector2_double

Synopsis: Multiply two vectors element wise

Declaration:
```
function operator *(Tvector2_double, Tvector2_double): Tvector2_double
                                                        (const x: Tvec
                                                        const y: Tvect
                                                         : Tvector2_do
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.30 operator *(Tvector2_extended, extended): Tvector2_extended

Synopsis: Multiply a two-dimensional extended precision vector by a scalar

Declaration:
```
function operator *(Tvector2_extended, extended): Tvector2_extended
                                                  (const x: Tvector
                                                  y: extended)
                                                   : Tvector2_exten
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.31 operator *(Tvector2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Multiply two vectors element wise

Declaration:
```
                                                  function
                                                  (const x
                                                  const y:
                                                   : Tvect
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.32 operator *(Tvector2_single, single): Tvector2_single

Synopsis: Multiply a two-dimensional single precision vector by a scalar

Declaration:
```
function operator *(Tvector2_single, single): Tvector2_single
                                              (const x: Tvector2_sing
                                              y: single)
                                               : Tvector2_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.33 operator *(Tvector2_single, Tvector2_single): Tvector2_single

Synopsis: Multiply two vectors element wise

Declaration:
```
function operator *(Tvector2_single, Tvector2_single): Tvector2_single
                                                       (const x: Tvec
                                                       const y: Tvect
                                                        : Tvector2_si
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.34 operator *(Tvector3_double, double): Tvector3_double

Synopsis: Multiply a three-dimensional double precision vector by a scalar

Declaration:
```
function operator *(Tvector3_double, double): Tvector3_double
                                                (const x: Tvector3_doub
                                                y: double)
                                                 : Tvector3_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.35 operator *(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Multiply two vectors element wise

Declaration:
```
function operator *(Tvector3_double, Tvector3_double): Tvector3_double
                                                         (const x: Tvec
                                                         const y: Tvect
                                                          : Tvector3_do
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.36 operator *(Tvector3_extended, extended): Tvector3_extended

Synopsis: Multiply a three-dimensional extended precision vector by a scalar

Declaration:
```
function operator *(Tvector3_extended, extended): Tvector3_extended
                                                    (const x: Tvector
                                                    y: extended)
                                                     : Tvector3_exten
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.37 operator *(Tvector3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Multiply two vectors element wise

Declaration:
```
                                                          function
                                                          (const x
                                                          const y:
                                                           : Tvect
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.38 operator *(Tvector3_single, single): Tvector3_single

Synopsis: Multiply a three-dimensional single precision vector by a scalar

Declaration:
```
function operator *(Tvector3_single, single): Tvector3_single
                                                (const x: Tvector3_sing
                                                y: single)
                                                 : Tvector3_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.39 operator *(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Multiply two vectors element wise

Declaration:
```
function operator *(Tvector3_single, Tvector3_single): Tvector3_single
                                                         (const x: Tvec
                                                         const y: Tvect
                                                          : Tvector3_si
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.40 operator *(Tvector4_double, double): Tvector4_double

Synopsis: Multiply a four-dimensional double precision vector by a scalar

Declaration:
```
function operator *(Tvector4_double, double): Tvector4_double
                                                (const x: Tvector4_doub
                                                y: double)
                                                 : Tvector4_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.41 operator *(Tvector4_double, Tvector4_double): Tvector4_double

Synopsis: Multiply two vectors element wise

Declaration:
```
function operator *(Tvector4_double, Tvector4_double): Tvector4_double
                                                         (const x: Tvec
                                                         const y: Tvect
                                                          : Tvector4_do
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.42 operator *(Tvector4_extended, extended): Tvector4_extended

Synopsis: Multiply a four-dimensional extended precision vector by a scalar

Declaration:
```
function operator *(Tvector4_extended, extended): Tvector4_extended
                                                    (const x: Tvector
                                                    y: extended)
                                                     : Tvector4_exten
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.43 operator *(Tvector4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Multiply two vectors element wise

Declaration:
```
                                                    function
                                                    (const x
                                                    const y:
                                                     : Tvect
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.44 operator *(Tvector4_single, single): Tvector4_single

Synopsis: Multiply a four-dimensional single precision vector by a scalar

Declaration:
```
function operator *(Tvector4_single, single): Tvector4_single
                                                  (const x: Tvector4_sing
                                                  y: single)
                                                   : Tvector4_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

18.3.45 operator *(Tvector4_single, Tvector4_single): Tvector4_single

Synopsis: Multiply two vectors element wise

Declaration:
```
function operator *(Tvector4_single, Tvector4_single): Tvector4_single
                                                    (const x: Tvec
                                                    const y: Tvect
                                                     : Tvector4_si
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

18.3.46 operator **(Tvector2_double, Tvector2_double): double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector2_double, Tvector2_double): double
                                                        (const x: Tvector2_doul
                                                        const y: Tvector2_doub
                                                         : double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.47 operator **(Tvector2_extended, Tvector2_extended): extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector2_extended, Tvector2_extended): extended
                                                          (const x: Tvecto
                                                          const y: Tvector
                                                           : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.48 operator **(Tvector2_single, Tvector2_single): single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector2_single, Tvector2_single): single
                                                        (const x: Tvector2_sin
                                                        const y: Tvector2_sing
                                                         : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.49 operator **(Tvector3_double, Tvector3_double): double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector3_double, Tvector3_double): double
                                                        (const x: Tvector3_doul
                                                        const y: Tvector3_doub
                                                         : double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.50 operator **(Tvector3_extended, Tvector3_extended): extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector3_extended, Tvector3_extended): extended
                                                          (const x: Tvecto
                                                          const y: Tvector
                                                           : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.51 operator **(Tvector3_single, Tvector3_single): single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector3_single, Tvector3_single): single
                                                        (const x: Tvector3_sin
                                                        const y: Tvector3_sing
                                                         : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.52 operator **(Tvector4_double, Tvector4_double): double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector4_double, Tvector4_double): double
                                                        (const x: Tvector4_dou
                                                        const y: Tvector4_doub
                                                         : double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.53 operator **(Tvector4_extended, Tvector4_extended): extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector4_extended, Tvector4_extended): extended
                                                          (const x: Tvecto
                                                          const y: Tvector
                                                           : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.54 operator **(Tvector4_single, Tvector4_single): single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
function operator **(Tvector4_single, Tvector4_single): single
                                                          (const x: Tvector4_sin
                                                          const y: Tvector4_sing
                                                           : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

18.3.55 operator +(Tmatrix2_double, double): Tmatrix2_double

Synopsis: Add scalar to two-dimensional double precision matrix

Declaration:
```
function operator +(Tmatrix2_double, double): Tmatrix2_double
                                                        (const m: Tmatrix2_doub
                                                        const x: double)
                                                         : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.56 operator +(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double

Synopsis: Add two two-dimensional double precision matrices together.

Declaration:
```
function operator +(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double
                                                              (const m1: Tma
                                                              const m2: Tmat
                                                               : Tmatrix2_do
```

Visibility: default

Description: This operator allows you to add two two-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.57 operator +(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix

Declaration:
```
function operator +(Tmatrix2_extended, extended): Tmatrix2_extended
                                                            (const m: Tmatrix
                                                            const x: extended
                                                             : Tmatrix2_exten
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.58 operator +(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended

Synopsis: Add two two-dimensional extended precision matrices together.

Declaration:

```
function
(const m
const m2
 : Tmatr
```

Visibility: default

Description: This operator allows you to add two two-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.59 operator +(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Add scalar to two-dimensional single precision matrix

Declaration:

```
function operator +(Tmatrix2_single, single): Tmatrix2_single
                                              (const m: Tmatrix2_sing
                                              const x: single)
                                               : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.60 operator +(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single

Synopsis: Add two two-dimensional single precision matrices together.

Declaration:

```
function operator +(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single
                                                        (const m1: Tma
                                                        const m2: Tmat
                                                         : Tmatrix2_si
```

Visibility: default

Description: This operator allows you to add two two-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.61 operator +(Tmatrix3_double, double): Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix

Declaration:

```
function operator +(Tmatrix3_double, double): Tmatrix3_double
                                              (const m: Tmatrix3_doub
                                              const x: double)
                                               : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.62 operator +(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double

Synopsis: Add two three-dimensional double precision matrices together.

Declaration:
```
function operator +(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double
                                                        (const m1: Tma
                                                        const m2: Tmat
                                                         : Tmatrix3_do
```

Visibility: default

Description: This operator allows you to add two three-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.63 operator +(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix

Declaration:
```
function operator +(Tmatrix3_extended, extended): Tmatrix3_extended
                                                   (const m: Tmatrix
                                                   const x: extended
                                                    : Tmatrix3_exten
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.64 operator +(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended

Synopsis: Add two three-dimensional extended precision matrices together.

Declaration:
```
                                                   function
                                                   (const m
                                                   const m2
                                                    : Tmatr
```

Visibility: default

Description: This operator allows you to add two three-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.65 operator +(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix

Declaration:
```
function operator +(Tmatrix3_single, single): Tmatrix3_single
                                               (const m: Tmatrix3_sing
                                               const x: single)
                                                : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.66 operator +(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single

Synopsis: Add two three-dimensional single precision matrices together.

Declaration:
```
function operator +(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single
                                                        (const m1: Tma
                                                        const m2: Tmat
                                                         : Tmatrix3_si
```

Visibility: default

Description: This operator allows you to add two three-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.67 operator +(Tmatrix4_double, double): Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix

Declaration:
```
function operator +(Tmatrix4_double, double): Tmatrix4_double
                                              (const m: Tmatrix4_doub
                                              const x: double)
                                               : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.68 operator +(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double

Synopsis: Add two four-dimensional double precision matrices together.

Declaration:
```
function operator +(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double
                                                        (const m1: Tma
                                                        const m2: Tmat
                                                         : Tmatrix4_do
```

Visibility: default

Description: This operator allows you to add two four-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.69 operator +(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix

Declaration:
```
function operator +(Tmatrix4_extended, extended): Tmatrix4_extended
                                                  (const m: Tmatrix
                                                  const x: extended
                                                   : Tmatrix4_exten
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.70 operator +(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended

Synopsis: Add two four-dimensional extended precision matrices together.

Declaration:

```
function
(const m
const m2
 : Tmatr
```

Visibility: default

Description: This operator allows you to add two four-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.71 operator +(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix

Declaration:

```
function operator +(Tmatrix4_single, single): Tmatrix4_single
                                              (const m: Tmatrix4_singl
                                              const x: single)
                                               : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

18.3.72 operator +(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single

Synopsis: Add two four-dimensional single precision matrices together.

Declaration:

```
function operator +(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single
                                                       (const m1: Tmat
                                                       const m2: Tmatr
                                                        : Tmatrix4_sin
```

Visibility: default

Description: This operator allows you to add two four-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

18.3.73 operator +(Tvector2_double, double): Tvector2_double

Synopsis: Add scalar to two-dimensional double precision vector

Declaration:

```
function operator +(Tvector2_double, double): Tvector2_double
                                              (const x: Tvector2_doubl
                                              y: double)
                                               : Tvector2_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.74 operator +(Tvector2_double, Tvector2_double): Tvector2_double

Synopsis: Add two-dimensional double precision vectors together

Declaration:
```
function operator +(Tvector2_double, Tvector2_double): Tvector2_double
                                                        (const x: Tvec
                                                        const y: Tvect
                                                         : Tvector2_do
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.75 operator +(Tvector2_extended, extended): Tvector2_extended

Synopsis: Add scalar to two-dimensional extended precision vector

Declaration:
```
function operator +(Tvector2_extended, extended): Tvector2_extended
                                                   (const x: Tvector
                                                   y: extended)
                                                    : Tvector2_exten
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.76 operator +(Tvector2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Add two-dimensional extended precision vectors together

Declaration:
```
                                                   function
                                                   (const x
                                                   const y:
                                                    : Tvect
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.77 operator +(Tvector2_single, single): Tvector2_single

Synopsis: Add scalar to two-dimensional single precision vector

Declaration:
```
function operator +(Tvector2_single, single): Tvector2_single
                                               (const x: Tvector2_sing
                                               y: single)
                                                : Tvector2_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.78 operator +(Tvector2_single, Tvector2_single): Tvector2_single

Synopsis: Add two-dimensional single precision vectors together

Declaration:
```
function operator +(Tvector2_single, Tvector2_single): Tvector2_single
                                                                (const x: Tvec
                                                                const y: Tvect
                                                                 : Tvector2_si
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.79 operator +(Tvector3_double, double): Tvector3_double

Synopsis: Add scalar to three-dimensional double precision vector

Declaration:
```
function operator +(Tvector3_double, double): Tvector3_double
                                                          (const x: Tvector3_doub
                                                          y: double)
                                                           : Tvector3_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.80 operator +(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Add three-dimensional double precision vectors together

Declaration:
```
function operator +(Tvector3_double, Tvector3_double): Tvector3_double
                                                                (const x: Tvec
                                                                const y: Tvect
                                                                 : Tvector3_do
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.81 operator +(Tvector3_extended, extended): Tvector3_extended

Synopsis: Add scalar to three-dimensional extended precision vector

Declaration:
```
function operator +(Tvector3_extended, extended): Tvector3_extended
                                                            (const x: Tvector
                                                            y: extended)
                                                             : Tvector3_exten
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.82 operator +(Tvector3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Add three-dimensional extended precision vectors together

Declaration:

```
function
(const x
const y:
 : Tvect
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.83 operator +(Tvector3_single, single): Tvector3_single

Synopsis: Add scalar to three-dimensional single precision vector

Declaration:

```
function operator +(Tvector3_single, single): Tvector3_single
                                                (const x: Tvector3_sing
                                                y: single)
                                                 : Tvector3_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.84 operator +(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Add three-dimensional extended precision vectors together

Declaration:

```
function operator +(Tvector3_single, Tvector3_single): Tvector3_single
                                                      (const x: Tvec
                                                      const y: Tvect
                                                       : Tvector3_si
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.85 operator +(Tvector4_double, double): Tvector4_double

Synopsis: Add scalar to four-dimensional double precision vector

Declaration:

```
function operator +(Tvector4_double, double): Tvector4_double
                                                (const x: Tvector4_doub
                                                y: double)
                                                 : Tvector4_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.86 operator +(Tvector4_double, Tvector4_double): Tvector4_double

Synopsis: Add four-dimensional double precision vectors together

Declaration:
```
function operator +(Tvector4_double, Tvector4_double): Tvector4_double
                                                                    (const x: Tvec
                                                                    const y: Tvect
                                                                     : Tvector4_do
```

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.87 operator +(Tvector4_extended, extended): Tvector4_extended

Synopsis: Add scalar to four-dimensional extended precision vector

Declaration:
```
function operator +(Tvector4_extended, extended): Tvector4_extended
                                                                  (const x: Tvector
                                                                  y: extended)
                                                                   : Tvector4_exten
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.88 operator +(Tvector4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Add four-dimensional extended precision vectors together

Declaration:
```
                                                                  function
                                                                  (const x
                                                                  const y:
                                                                   : Tvect
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.89 operator +(Tvector4_single, single): Tvector4_single

Synopsis: Add scalar to four-dimensional single precision vector

Declaration:
```
function operator +(Tvector4_single, single): Tvector4_single
                                                                (const x: Tvector4_sing
                                                                y: single)
                                                                 : Tvector4_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

18.3.90 operator +(Tvector4_single, Tvector4_single): Tvector4_single

Synopsis: Add four-dimensional single precision vectors together

Declaration:
```
function operator +(Tvector4_single, Tvector4_single): Tvector4_single
                                                                    (const x: Tvec
                                                                    const y: Tvect
                                                                     : Tvector4_si
```

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

18.3.91 operator -(Tmatrix2_double): Tmatrix2_double

Synopsis: Negate two-dimensional double precision matrix.

Declaration:
```
function operator -(Tmatrix2_double): Tmatrix2_double
                                                  (const m1: Tmatrix2_double)
                                                   : Tmatrix2_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.92 operator -(Tmatrix2_double, double): Tmatrix2_double

Synopsis: Subtract scalar to two-dimensional double precision matrix

Declaration:
```
function operator -(Tmatrix2_double, double): Tmatrix2_double
                                                        (const m: Tmatrix2_doub
                                                        const x: double)
                                                         : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.93 operator -(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double

Synopsis: Subtract a two-dimensional double precision matrix from another.

Declaration:
```
function operator -(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double
                                                                    (const m1: Tma
                                                                    const m2: Tmat
                                                                     : Tmatrix2_do
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.94 operator -(Tmatrix2_extended): Tmatrix2_extended

Synopsis: Negate two-dimensional extended precision matrix.

Declaration:
```
function operator -(Tmatrix2_extended): Tmatrix2_extended
                                                    (const m1: Tmatrix2_extende
                                                     : Tmatrix2_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.95 operator -(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix

Declaration:
```
function operator -(Tmatrix2_extended, extended): Tmatrix2_extended
                                                          (const m: Tmatrix
                                                          const x: extended
                                                           : Tmatrix2_exten
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.96 operator -(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended

Synopsis: Subtract a two-dimensional extended precision matrix from another.

Declaration:
```
                                                                  function
                                                                  (const m
                                                                  const m2
                                                                   : Tmatr
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.97 operator -(Tmatrix2_single): Tmatrix2_single

Synopsis: Negate two-dimensional single precision matrix.

Declaration:
```
function operator -(Tmatrix2_single): Tmatrix2_single
                                                 (const m1: Tmatrix2_single)
                                                  : Tmatrix2_single
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.98 operator -(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Subtract scalar to two-dimensional single precision matrix

Declaration:
```
function operator -(Tmatrix2_single, single): Tmatrix2_single
                                                (const m: Tmatrix2_sing
                                                const x: single)
                                                 : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.99 operator -(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single

Synopsis: Subtract a two-dimensional single precision matrix from another.

Declaration:
```
function operator -(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single
                                                         (const m1: Tma
                                                         const m2: Tmat
                                                          : Tmatrix2_si
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.100 operator -(Tmatrix3_double): Tmatrix3_double

Synopsis: Negate three-dimensional double precision matrix.

Declaration:
```
function operator -(Tmatrix3_double): Tmatrix3_double
                                        (const m1: Tmatrix3_double)
                                         : Tmatrix3_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.101 operator -(Tmatrix3_double, double): Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix

Declaration:
```
function operator -(Tmatrix3_double, double): Tmatrix3_double
                                                (const m: Tmatrix3_doub
                                                const x: double)
                                                 : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.102 operator -(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double

Synopsis: Subtract a three-dimensional double precision matrix from another.

Declaration:
```
function operator -(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double
                                                                    (const m1: Tma
                                                                    const m2: Tmat
                                                                     : Tmatrix3_do
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.103 operator -(Tmatrix3_extended): Tmatrix3_extended

Synopsis: Negate three-dimensional extended precision matrix.

Declaration:
```
function operator -(Tmatrix3_extended): Tmatrix3_extended
                                                  (const m1: Tmatrix3_extende
                                                   : Tmatrix3_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.104 operator -(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix

Declaration:
```
function operator -(Tmatrix3_extended, extended): Tmatrix3_extended
                                                        (const m: Tmatrix
                                                        const x: extended
                                                         : Tmatrix3_exten
```

Visibility: default

Description: This operator allows you to substract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.105 operator -(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended

Synopsis: Subtract a three-dimensional extended precision matrix from another.

Declaration:
```
                                                                    function
                                                                    (const m
                                                                    const m2
                                                                     : Tmatr
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.106 operator -(Tmatrix3_single): Tmatrix3_single

Synopsis: Negate three-dimensional single precision matrix.

Declaration:
```
function operator -(Tmatrix3_single): Tmatrix3_single
                                                    (const m1: Tmatrix3_single)
                                                     : Tmatrix3_single
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.107 operator -(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix

Declaration:
```
function operator -(Tmatrix3_single, single): Tmatrix3_single
                                                          (const m: Tmatrix3_sing
                                                          const x: single)
                                                           : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.108 operator -(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single

Synopsis: Subtract a three-dimensional single precision matrix from another.

Declaration:
```
function operator -(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single
                                                                (const m1: Tma
                                                                const m2: Tmat
                                                                 : Tmatrix3_si
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.109 operator -(Tmatrix4_double): Tmatrix4_double

Synopsis: Negate four-dimensional double precision matrix.

Declaration:
```
function operator -(Tmatrix4_double): Tmatrix4_double
                                                    (const m1: Tmatrix4_double)
                                                     : Tmatrix4_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.110 operator -(Tmatrix4_double, double): Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix

Declaration:
```
function operator -(Tmatrix4_double, double): Tmatrix4_double
                                                  (const m: Tmatrix4_doub
                                                  const x: double)
                                                   : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.111 operator -(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double

Synopsis: Subtract a four-dimensional double precision matrix from another.

Declaration:
```
function operator -(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double
                                                        (const m1: Tma
                                                        const m2: Tmat
                                                         : Tmatrix4_do
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.112 operator -(Tmatrix4_extended): Tmatrix4_extended

Synopsis: Negate four-dimensional extended precision matrix.

Declaration:
```
function operator -(Tmatrix4_extended): Tmatrix4_extended
                                          (const m1: Tmatrix4_extende
                                           : Tmatrix4_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.113 operator -(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix

Declaration:
```
function operator -(Tmatrix4_extended, extended): Tmatrix4_extended
                                                    (const m: Tmatrix
                                                    const x: extended
                                                     : Tmatrix4_exten
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.114 operator -(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended

Synopsis: Subtract a four-dimensional extended precision matrix from another.

Declaration:

```
function
(const m
const m2
 : Tmatr
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.115 operator -(Tmatrix4_single): Tmatrix4_single

Synopsis: Negate four-dimensional single precision matrix.

Declaration:

```
function operator -(Tmatrix4_single): Tmatrix4_single
                                      (const m1: Tmatrix4_single)
                                       : Tmatrix4_single
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

18.3.116 operator -(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix

Declaration:

```
function operator -(Tmatrix4_single, single): Tmatrix4_single
                                              (const m: Tmatrix4_sing
                                              const x: single)
                                               : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

18.3.117 operator -(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single

Synopsis: Subtract a four-dimensional single precision matrix from another.

Declaration:

```
function operator -(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single
                                                       (const m1: Tma
                                                       const m2: Tmat
                                                        : Tmatrix4_si
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

18.3.118 operator -(Tvector2_double): Tvector2_double

Synopsis: Negate two-dimensional vector.

Declaration:
```
function operator -(Tvector2_double): Tvector2_double
                                                    (const x: Tvector2_double)
                                                     : Tvector2_double
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.119 operator -(Tvector2_double, double): Tvector2_double

Synopsis: Subtract scalar from two-dimensional double precision vector

Declaration:
```
function operator -(Tvector2_double, double): Tvector2_double
                                                          (const x: Tvector2_doub
                                                          y: double)
                                                           : Tvector2_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.120 operator -(Tvector2_double, Tvector2_double): Tvector2_double

Synopsis: Subtract two-dimensional double precision vectors from each other

Declaration:
```
function operator -(Tvector2_double, Tvector2_double): Tvector2_double
                                                                (const x: Tvec
                                                                const y: Tvect
                                                                 : Tvector2_do
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.121 operator -(Tvector2_extended): Tvector2_extended

Synopsis: Negate two-dimensional vector.

Declaration:
```
function operator -(Tvector2_extended): Tvector2_extended
                                                       (const x: Tvector2_extended
                                                        : Tvector2_extended
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.122 operator -(Tvector2_extended, extended): Tvector2_extended

Synopsis: Subtract scalar from two-dimensional extended precision vector

Declaration:
```
function operator -(Tvector2_extended, extended): Tvector2_extended
                                                    (const x: Tvector
                                                    y: extended)
                                                     : Tvector2_exten
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.123 operator -(Tvector2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Subtract two-dimensional extended precision vectors from each other

Declaration:
```
                                                    function
                                                    (const x
                                                    const y:
                                                     : Tvect
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.124 operator -(Tvector2_single): Tvector2_single

Synopsis: Negate two-dimensional vector.

Declaration:
```
function operator -(Tvector2_single): Tvector2_single
                                        (const x: Tvector2_single)
                                         : Tvector2_single
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.125 operator -(Tvector2_single, single): Tvector2_single

Synopsis: Subtract scalar from two-dimensional single precision vector

Declaration:
```
function operator -(Tvector2_single, single): Tvector2_single
                                                (const x: Tvector2_sing
                                                y: single)
                                                 : Tvector2_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.126 operator -(Tvector2_single, Tvector2_single): Tvector2_single

Synopsis: Subtract two-dimensional single precision vectors from each other

Declaration:
```
function operator -(Tvector2_single, Tvector2_single): Tvector2_single
                                                                    (const x: Tvec
                                                                    const y: Tvect
                                                                     : Tvector2_si
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.127 operator -(Tvector3_double): Tvector3_double

Synopsis: Negate three-dimensional vector.

Declaration:
```
function operator -(Tvector3_double): Tvector3_double
                                                  (const x: Tvector3_double)
                                                   : Tvector3_double
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.128 operator -(Tvector3_double, double): Tvector3_double

Synopsis: Subtract scalar from three-dimensional double precision vector

Declaration:
```
function operator -(Tvector3_double, double): Tvector3_double
                                                        (const x: Tvector3_doub
                                                        y: double)
                                                         : Tvector3_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.129 operator -(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Subtract three-dimensional double precision vectors from each other

Declaration:
```
function operator -(Tvector3_double, Tvector3_double): Tvector3_double
                                                                    (const x: Tvec
                                                                    const y: Tvect
                                                                     : Tvector3_do
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.130 operator -(Tvector3_extended): Tvector3_extended

Synopsis: Negate three-dimensional vector.

Declaration:
```
function operator -(Tvector3_extended): Tvector3_extended
                                                    (const x: Tvector3_extended
                                                     : Tvector3_extended
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.131 operator -(Tvector3_extended, extended): Tvector3_extended

Synopsis: Subtract scalar from three-dimensional extended precision vector

Declaration:
```
function operator -(Tvector3_extended, extended): Tvector3_extended
                                                          (const x: Tvector
                                                          y: extended)
                                                           : Tvector3_exten
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.132 operator -(Tvector3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Subtract three-dimensional extended precision vectors from each other

Declaration:
```
                                                                  function
                                                                  (const x
                                                                  const y:
                                                                   : Tvect
```

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.133 operator -(Tvector3_single): Tvector3_single

Synopsis: Negate three-dimensional vector.

Declaration:
```
function operator -(Tvector3_single): Tvector3_single
                                                 (const x: Tvector3_single)
                                                  : Tvector3_single
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.134 operator -(Tvector3_single, single): Tvector3_single

Synopsis: Subtract scalar from three-dimensional single precision vector

Declaration:
```
function operator -(Tvector3_single, single): Tvector3_single
                                                    (const x: Tvector3_sing
                                                    y: single)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.135 operator -(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Subtract three-dimensional single precision vectors from each other

Declaration:
```
function operator -(Tvector3_single, Tvector3_single): Tvector3_single
                                                          (const x: Tvec
                                                          const y: Tvect
                                                           : Tvector3_si
```

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.136 operator -(Tvector4_double): Tvector4_double

Synopsis: Negate four-dimensional vector.

Declaration:
```
function operator -(Tvector4_double): Tvector4_double
                                          (const x: Tvector4_double)
                                           : Tvector4_double
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.137 operator -(Tvector4_double, double): Tvector4_double

Synopsis: Subtract scalar from four-dimensional double precision vector

Declaration:
```
function operator -(Tvector4_double, double): Tvector4_double
                                                    (const x: Tvector4_doub
                                                    y: double)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.138 operator -(Tvector4_double, Tvector4_double): Tvector4_double

Synopsis: Subtract four-dimensional double precision vectors from each other

Declaration:
```
function operator -(Tvector4_double, Tvector4_double): Tvector4_double
                                                                    (const x: Tvec
                                                                    const y: Tvect
                                                                     : Tvector4_do
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.139 operator -(Tvector4_extended): Tvector4_extended

Synopsis: Negate four-dimensional vector.

Declaration:
```
function operator -(Tvector4_extended): Tvector4_extended
                                                  (const x: Tvector4_extended
                                                   : Tvector4_extended
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.140 operator -(Tvector4_extended, extended): Tvector4_extended

Synopsis: Subtract scalar from four-dimensional extended precision vector

Declaration:
```
function operator -(Tvector4_extended, extended): Tvector4_extended
                                                        (const x: Tvector
                                                       y: extended)
                                                        : Tvector4_exten
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.141 operator -(Tvector4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Subtract four-dimensional extended precision vectors from each other

Declaration:
```
                                                                    function
                                                                    (const x
                                                                    const y:
                                                                     : Tvect
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.142 operator -(Tvector4_single): Tvector4_single

Synopsis: Negate four-dimensional vector.

Declaration:
```
function operator -(Tvector4_single): Tvector4_single
                                                    (const x: Tvector4_single)
                                                     : Tvector4_single
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

18.3.143 operator -(Tvector4_single, single): Tvector4_single

Synopsis: Subtract scalar from four-dimensional single precision vector

Declaration:
```
function operator -(Tvector4_single, single): Tvector4_single
                                                          (const x: Tvector4_sing
                                                          y: single)
                                                           : Tvector4_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

18.3.144 operator -(Tvector4_single, Tvector4_single): Tvector4_single

Synopsis: Subtract four-dimensional single precision vectors from each other

Declaration:
```
function operator -(Tvector4_single, Tvector4_single): Tvector4_single
                                                                (const x: Tvec
                                                                const y: Tvect
                                                                 : Tvector4_si
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

18.3.145 operator /(Tmatrix2_double, double): Tmatrix2_double

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix2_double, double): Tmatrix2_double
                                                          (const m: Tmatrix2_doub
                                                          const x: double)
                                                           : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.146 operator /(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix2_extended, extended): Tmatrix2_extended
                                                  (const m: Tmatrix
                                                  const x: extended
                                                   : Tmatrix2_exten
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.147 operator /(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix2_single, single): Tmatrix2_single
                                              (const m: Tmatrix2_sing
                                              const x: single)
                                               : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.148 operator /(Tmatrix3_double, double): Tmatrix3_double

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix3_double, double): Tmatrix3_double
                                              (const m: Tmatrix3_doub
                                              const x: double)
                                               : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.149 operator /(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix3_extended, extended): Tmatrix3_extended
                                                  (const m: Tmatrix
                                                  const x: extended
                                                   : Tmatrix3_exten
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.150 operator /(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix3_single, single): Tmatrix3_single
                                                    (const m: Tmatrix3_sing
                                                    const x: single)
                                                     : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.151 operator /(Tmatrix4_double, double): Tmatrix4_double

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix4_double, double): Tmatrix4_double
                                                    (const m: Tmatrix4_doub
                                                    const x: double)
                                                     : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.152 operator /(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix4_extended, extended): Tmatrix4_extended
                                                        (const m: Tmatrix
                                                        const x: extended
                                                         : Tmatrix4_exten
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.153 operator /(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Divide a two-dimensional single precision matrix bye a scalar

Declaration:
```
function operator /(Tmatrix4_single, single): Tmatrix4_single
                                                    (const m: Tmatrix4_sing
                                                    const x: single)
                                                     : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

18.3.154 operator /(Tvector2_double, double): Tvector2_double

Synopsis: Divide a two-dimensional double precision vector by a scalar

Declaration:
```
function operator /(Tvector2_double, double): Tvector2_double
                                                (const x: Tvector2_doub
                                                y: double)
                                                 : Tvector2_double
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.155 operator /(Tvector2_extended, extended): Tvector2_extended

Synopsis: Divide a two-dimensional extended precision vector by a scalar

Declaration:
```
function operator /(Tvector2_extended, extended): Tvector2_extended
                                                    (const x: Tvector
                                                    y: extended)
                                                     : Tvector2_exten
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.156 operator /(Tvector2_single, single): Tvector2_single

Synopsis: Divide a two-dimensional single precision vector by a scalar

Declaration:
```
function operator /(Tvector2_single, single): Tvector2_single
                                                (const x: Tvector2_sing
                                                y: single)
                                                 : Tvector2_single
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.157 operator /(Tvector3_double, double): Tvector3_double

Synopsis: Divide a three-dimensional double precision vector by a scalar

Declaration:
```
function operator /(Tvector3_double, double): Tvector3_double
                                                (const x: Tvector3_doub
                                                y: double)
                                                 : Tvector3_double
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.158 operator /(Tvector3_extended, extended): Tvector3_extended

Synopsis: Divide a three-dimensional extended precision vector by a scalar

Declaration:
```
function operator /(Tvector3_extended, extended): Tvector3_extended
                                                    (const x: Tvector
                                                    y: extended)
                                                     : Tvector3_exten
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.159 operator /(Tvector3_single, single): Tvector3_single

Synopsis: Divide a three-dimensional single precision vector by a scalar

Declaration:
```
function operator /(Tvector3_single, single): Tvector3_single
                                                (const x: Tvector3_sing
                                                y: single)
                                                 : Tvector3_single
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.160 operator /(Tvector4_double, double): Tvector4_double

Synopsis: Divide a four-dimensional double precision vector by a scalar

Declaration:
```
function operator /(Tvector4_double, double): Tvector4_double
                                                (const x: Tvector4_doub
                                                y: double)
                                                 : Tvector4_double
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.161 operator /(Tvector4_extended, extended): Tvector4_extended

Synopsis: Divide a four-dimensional extended precision vector by a scalar

Declaration:
```
function operator /(Tvector4_extended, extended): Tvector4_extended
                                                    (const x: Tvector
                                                    y: extended)
                                                     : Tvector4_exten
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.162 operator /(Tvector4_single, single): Tvector4_single

Synopsis: Divide a four-dimensional single precision vector by a scalar

Declaration:
```
function operator /(Tvector4_single, single): Tvector4_single
                                                    (const x: Tvector4_sing
                                                    y: single)
                                                     : Tvector4_single
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

18.3.163 operator :=(Tmatrix2_double): Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix2_extended
                                                 (const v: Tmatrix2_double)
                                                  : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected.

18.3.164 operator :=(Tmatrix2_double): Tmatrix2_single

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix2_single
                                                 (const v: Tmatrix2_double)
                                                  : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. Some accurace is lost because of the conversion.

18.3.165 operator :=(Tmatrix2_double): Tmatrix3_double

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix3_double
                                                 (const v: Tmatrix2_double)
                                                  : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

18.3.166 operator :=(Tmatrix2_double): Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix3_extended
                                                    (const v: Tmatrix2_double)
                                                     : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

18.3.167 operator :=(Tmatrix2_double): Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix3_single
                                                  (const v: Tmatrix2_double)
                                                   : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

18.3.168 operator :=(Tmatrix2_double): Tmatrix4_double

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix4_double
                                                  (const v: Tmatrix2_double)
                                                   : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

18.3.169 operator :=(Tmatrix2_double): Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix4_extended
                                                    (const v: Tmatrix2_double)
                                                     : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

18.3.170 operator :=(Tmatrix2_double): Tmatrix4_single

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_double): Tmatrix4_single
                                                    (const v: Tmatrix2_double)
                                                     : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

18.3.171 operator :=(Tmatrix2_extended): Tmatrix2_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix2_double
                                                      (const v: Tmatrix2_extended)
                                                       : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional two with extended precision values wherever a two-dimensional matrix with double precision is expected. Some accuracy is lost because of the conversion.

18.3.172 operator :=(Tmatrix2_extended): Tmatrix2_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix2_single
                                                      (const v: Tmatrix2_extended)
                                                       : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. Some accuracy is lost because of the conversion.

18.3.173 operator :=(Tmatrix2_extended): Tmatrix3_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix3_double
                                                      (const v: Tmatrix2_extended)
                                                       : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

18.3.174 operator :=(Tmatrix2_extended): Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix3_extended
                                                    (const v: Tmatrix2_extende
                                                     : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

18.3.175 operator :=(Tmatrix2_extended): Tmatrix3_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix3_single
                                                  (const v: Tmatrix2_extended)
                                                   : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

18.3.176 operator :=(Tmatrix2_extended): Tmatrix4_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix4_double
                                                  (const v: Tmatrix2_extended)
                                                   : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

18.3.177 operator :=(Tmatrix2_extended): Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix4_extended
                                                    (const v: Tmatrix2_extende
                                                     : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

18.3.178 operator :=(Tmatrix2_extended): Tmatrix4_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_extended): Tmatrix4_single
                                                    (const v: Tmatrix2_extended)
                                                     : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

18.3.179 operator :=(Tmatrix2_single): Tmatrix2_double

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix2_double
                                                  (const v: Tmatrix2_single)
                                                   : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected.

18.3.180 operator :=(Tmatrix2_single): Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix2_extended
                                                    (const v: Tmatrix2_single)
                                                     : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected.

18.3.181 operator :=(Tmatrix2_single): Tmatrix3_double

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix3_double
                                                  (const v: Tmatrix2_single)
                                                   : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

18.3.182 operator :=(Tmatrix2_single): Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix3_extended
                                                    (const v: Tmatrix2_single)
                                                     : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

18.3.183 operator :=(Tmatrix2_single): Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix3_single
                                                  (const v: Tmatrix2_single)
                                                   : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0.

18.3.184 operator :=(Tmatrix2_single): Tmatrix4_double

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix4_double
                                                  (const v: Tmatrix2_single)
                                                   : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

18.3.185 operator :=(Tmatrix2_single): Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix4_extended
                                                    (const v: Tmatrix2_single)
                                                     : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

18.3.186 operator :=(Tmatrix2_single): Tmatrix4_single

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix2_single): Tmatrix4_single
                                                    (const v: Tmatrix2_single)
                                                     : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

18.3.187 operator :=(Tmatrix3_double): Tmatrix2_double

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix2_double
                                                    (const v: Tmatrix3_double)
                                                     : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.188 operator :=(Tmatrix3_double): Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix2_extended
                                                      (const v: Tmatrix3_double)
                                                       : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

18.3.189 operator :=(Tmatrix3_double): Tmatrix2_single

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix2_single
                                                    (const v: Tmatrix3_double)
                                                     : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some accuracy is lost because of the conversion.

18.3.190 operator :=(Tmatrix3_double): Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix3_extended
                                                    (const v: Tmatrix3_double)
                                                     : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected.

18.3.191 operator :=(Tmatrix3_double): Tmatrix3_single

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix3_single
                                                  (const v: Tmatrix3_double)
                                                   : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

18.3.192 operator :=(Tmatrix3_double): Tmatrix4_double

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix4_double
                                                  (const v: Tmatrix3_double)
                                                   : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected.

18.3.193 operator :=(Tmatrix3_double): Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix4_extended
                                                    (const v: Tmatrix3_double)
                                                     : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

18.3.194 operator :=(Tmatrix3_double): Tmatrix4_single

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_double): Tmatrix4_single
                                                  (const v: Tmatrix3_double)
                                                   : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

18.3.195 operator :=(Tmatrix3_extended): Tmatrix2_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix2_double
                                                    (const v: Tmatrix3_extended)
                                                     : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some accuracy is lost becuase of the conversion.

18.3.196 operator :=(Tmatrix3_extended): Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix2_extended
                                                      (const v: Tmatrix3_extende
                                                       : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

18.3.197 operator :=(Tmatrix3_extended): Tmatrix2_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix2_single
                                                    (const v: Tmatrix3_extended)
                                                     : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

18.3.198 operator :=(Tmatrix3_extended): Tmatrix3_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix3_double
                                                    (const v: Tmatrix3_extended)
                                                     : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

18.3.199 operator :=(Tmatrix3_extended): Tmatrix3_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix3_single
                                                    (const v: Tmatrix3_extended)
                                                     : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

18.3.200 operator :=(Tmatrix3_extended): Tmatrix4_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix4_double
                                                    (const v: Tmatrix3_extended)
                                                     : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

18.3.201 operator :=(Tmatrix3_extended): Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix4_extended
                                                    (const v: Tmatrix3_extende
                                                     : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with extended precision is expected.

18.3.202 operator :=(Tmatrix3_extended): Tmatrix4_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_extended): Tmatrix4_single
                                                  (const v: Tmatrix3_extended)
                                                   : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

18.3.203 operator :=(Tmatrix3_single): Tmatrix2_double

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix2_double
                                                (const v: Tmatrix3_single)
                                                 : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.204 operator :=(Tmatrix3_single): Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix2_extended
                                                  (const v: Tmatrix3_single)
                                                   : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

18.3.205 operator :=(Tmatrix3_single): Tmatrix2_single

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix2_single
                                                    (const v: Tmatrix3_single)
                                                     : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

18.3.206 operator :=(Tmatrix3_single): Tmatrix3_double

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix3_double
                                                    (const v: Tmatrix3_single)
                                                     : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected.

18.3.207 operator :=(Tmatrix3_single): Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix3_extended
                                                      (const v: Tmatrix3_single)
                                                       : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected.

18.3.208 operator :=(Tmatrix3_single): Tmatrix4_double

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix4_double
                                                    (const v: Tmatrix3_single)
                                                     : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

18.3.209 operator :=(Tmatrix3_single): Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix4_extended
                                                    (const v: Tmatrix3_single)
                                                     : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

18.3.210 operator :=(Tmatrix3_single): Tmatrix4_single

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix3_single): Tmatrix4_single
                                                  (const v: Tmatrix3_single)
                                                   : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected.

18.3.211 operator :=(Tmatrix4_double): Tmatrix2_double

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix2_double
                                                  (const v: Tmatrix4_double)
                                                   : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.212 operator :=(Tmatrix4_double): Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix2_extended
                                                    (const v: Tmatrix4_double)
                                                     : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

18.3.213 operator :=(Tmatrix4_double): Tmatrix2_single

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix2_single
                                                    (const v: Tmatrix4_double)
                                                     : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

18.3.214 operator :=(Tmatrix4_double): Tmatrix3_double

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix3_double
                                                    (const v: Tmatrix4_double)
                                                     : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.215 operator :=(Tmatrix4_double): Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix3_extended
                                                      (const v: Tmatrix4_double)
                                                       : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

18.3.216 operator :=(Tmatrix4_double): Tmatrix3_single

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix3_single
                                                    (const v: Tmatrix4_double)
                                                     : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

18.3.217 operator :=(Tmatrix4_double): Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix4_extended
                                                    (const v: Tmatrix4_double)
                                                     : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

18.3.218 operator :=(Tmatrix4_double): Tmatrix4_single

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_double): Tmatrix4_single
                                                  (const v: Tmatrix4_double)
                                                   : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

18.3.219 operator :=(Tmatrix4_extended): Tmatrix2_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix2_double
                                                    (const v: Tmatrix4_extended)
                                                     : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

18.3.220 operator :=(Tmatrix4_extended): Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix2_extended
                                                      (const v: Tmatrix4_extende
                                                       : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

18.3.221 operator :=(Tmatrix4_extended): Tmatrix2_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix2_single
                                                    (const v: Tmatrix4_extended)
                                                     : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

18.3.222 operator :=(Tmatrix4_extended): Tmatrix3_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix3_double
                                                    (const v: Tmatrix4_extended)
                                                     : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.223 operator :=(Tmatrix4_extended): Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix3_extended
                                                      (const v: Tmatrix4_extende
                                                       : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.224 operator :=(Tmatrix4_extended): Tmatrix3_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix3_single
                                                    (const v: Tmatrix4_extended)
                                                     : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

18.3.225 operator :=(Tmatrix4_extended): Tmatrix4_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix4_double
                                                    (const v: Tmatrix4_extended)
                                                     : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected.

18.3.226 operator :=(Tmatrix4_extended): Tmatrix4_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_extended): Tmatrix4_single
                                                    (const v: Tmatrix4_extended)
                                                     : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

18.3.227 operator :=(Tmatrix4_single): Tmatrix2_double

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix2_double
                                                  (const v: Tmatrix4_single)
                                                   : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.228 operator :=(Tmatrix4_single): Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix2_extended
                                                    (const v: Tmatrix4_single)
                                                     : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

18.3.229 operator :=(Tmatrix4_single): Tmatrix2_single

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix2_single
                                                    (const v: Tmatrix4_single)
                                                     : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

18.3.230 operator :=(Tmatrix4_single): Tmatrix3_double

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix3_double
                                                    (const v: Tmatrix4_single)
                                                     : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

18.3.231 operator :=(Tmatrix4_single): Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix3_extended
                                                      (const v: Tmatrix4_single)
                                                       : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

18.3.232 operator :=(Tmatrix4_single): Tmatrix3_single

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional single precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix3_single
                                                    (const v: Tmatrix4_single)
                                                     : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away.

18.3.233 operator :=(Tmatrix4_single): Tmatrix4_double

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional double precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix4_double
                                                    (const v: Tmatrix4_single)
                                                     : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

18.3.234 operator :=(Tmatrix4_single): Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration:
```
function operator :=(Tmatrix4_single): Tmatrix4_extended
                                                      (const v: Tmatrix4_single)
                                                       : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

18.3.235 operator :=(Tvector2_double): Tvector2_extended

Synopsis: Allow assignment of double precision vector to extended precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector2_extended
                                                      (const v: Tvector2_double)
                                                       : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever an extended precision vector is expected.

18.3.236 operator :=(Tvector2_double): Tvector2_single

Synopsis: Allow assignment of double precision vector to single precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector2_single
                                                    (const v: Tvector2_double)
                                                     : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever a single precision vector is expected, at the cost of loosing some precision.

18.3.237 operator :=(Tvector2_double): Tvector3_double

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector3_double
                                                    (const v: Tvector2_double)
                                                     : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

18.3.238 operator :=(Tvector2_double): Tvector3_extended

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector3_extended
                                                      (const v: Tvector2_double)
                                                       : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

18.3.239 operator :=(Tvector2_double): Tvector3_single

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional single precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector3_single
                                                    (const v: Tvector2_double)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

18.3.240 operator :=(Tvector2_double): Tvector4_double

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector4_double
                                                    (const v: Tvector2_double)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

18.3.241 operator :=(Tvector2_double): Tvector4_extended

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector4_extended
                                                    (const v: Tvector2_double)
                                                     : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

18.3.242 operator :=(Tvector2_double): Tvector4_single

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector2_double): Tvector4_single
                                                  (const v: Tvector2_double)
                                                   : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accurace is lost because of the conversion and the third and fourth dimensions are set to 0.0.

18.3.243 operator :=(Tvector2_extended): Tvector2_double

Synopsis: Allow assignment of extended precision vector to double precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector2_double
                                                    (const v: Tvector2_extended)
                                                     : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a double precision vector is expected, at the cost of loosing some precision.

18.3.244 operator :=(Tvector2_extended): Tvector2_single

Synopsis: Allow assignment of extended precision vector to single precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector2_single
                                                    (const v: Tvector2_extended)
                                                     : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a single precision vector is expected, at the cost of loosing some precision.

18.3.245 operator :=(Tvector2_extended): Tvector3_double

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector3_double
                                                    (const v: Tvector2_extended)
                                                     : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

18.3.246 operator :=(Tvector2_extended): Tvector3_extended

Synopsis: Allow assignment of two-dimensional extened precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector3_extended
                                                      (const v: Tvector2_extende
                                                       : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

18.3.247 operator :=(Tvector2_extended): Tvector3_single

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional single precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector3_single
                                                    (const v: Tvector2_extended)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

18.3.248 operator :=(Tvector2_extended): Tvector4_double

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector4_double
                                                    (const v: Tvector2_extended)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

18.3.249 operator :=(Tvector2_extended): Tvector4_extended

Synopsis: Allow assignment of two-dimensiona extended precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector4_extended
                                                    (const v: Tvector2_extende
                                                     : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

18.3.250 operator :=(Tvector2_extended): Tvector4_single

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector2_extended): Tvector4_single
                                                  (const v: Tvector2_extended)
                                                   : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

18.3.251 operator :=(Tvector2_single): Tvector2_double

Synopsis: Allow assignment of single precision vector to double precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector2_double
                                                (const v: Tvector2_single)
                                                 : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever a double precision vector is expected.

18.3.252 operator :=(Tvector2_single): Tvector2_extended

Synopsis: Allow assignment of single precision vector to extended precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector2_extended
                                                  (const v: Tvector2_single)
                                                   : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever an extended precision vector is expected.

18.3.253 operator :=(Tvector2_single): Tvector3_double

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector3_double
                                                    (const v: Tvector2_single)
                                                     : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

18.3.254 operator :=(Tvector2_single): Tvector3_extended

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector3_extended
                                                      (const v: Tvector2_single)
                                                       : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

18.3.255 operator :=(Tvector2_single): Tvector3_single

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional single precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector3_single
                                                    (const v: Tvector2_single)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The third dimension is set to 0.0.

18.3.256 operator :=(Tvector2_single): Tvector4_double

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector4_double
                                                    (const v: Tvector2_single)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

18.3.257 operator :=(Tvector2_single): Tvector4_extended

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector4_extended
                                                    (const v: Tvector2_single)
                                                     : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

18.3.258 operator :=(Tvector2_single): Tvector4_single

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector2_single): Tvector4_single
                                                  (const v: Tvector2_single)
                                                   : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The third and fourth dimensions are set to 0.0.

18.3.259 operator :=(Tvector3_double): Tvector2_double

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector2_double
                                                  (const v: Tvector3_double)
                                                   : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

18.3.260 operator :=(Tvector3_double): Tvector2_extended

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector2_extended
                                                    (const v: Tvector3_double)
                                                     : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

18.3.261 operator :=(Tvector3_double): Tvector2_single

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional single precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector2_single
                                                    (const v: Tvector3_double)
                                                     : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

18.3.262 operator :=(Tvector3_double): Tvector3_extended

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector3_extended
                                                      (const v: Tvector3_double)
                                                       : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected.

18.3.263 operator :=(Tvector3_double): Tvector3_single

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional single precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector3_single
                                                    (const v: Tvector3_double)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some precision is lost because of the conversion.

18.3.264 operator :=(Tvector3_double): Tvector4_double

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector4_double
                                                    (const v: Tvector3_double)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

18.3.265 operator :=(Tvector3_double): Tvector4_extended

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector4_extended
                                                    (const v: Tvector3_double)
                                                     : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

18.3.266 operator :=(Tvector3_double): Tvector4_single

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector3_double): Tvector4_single
                                                  (const v: Tvector3_double)
                                                   : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some precision is lost because of the conversion.

18.3.267 operator :=(Tvector3_extended): Tvector2_double

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector2_double
                                                    (const v: Tvector3_extended)
                                                     : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

18.3.268 operator :=(Tvector3_extended): Tvector2_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector2_extended
                                                      (const v: Tvector3_extende
                                                       : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

18.3.269 operator :=(Tvector3_extended): Tvector2_single

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional single precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector2_single
                                                    (const v: Tvector3_extended)
                                                     : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

18.3.270 operator :=(Tvector3_extended): Tvector3_double

Synopsis: Allow assignment of three-dimensional extended precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector3_double
                                                    (const v: Tvector3_extended)
                                                     : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

18.3.271 operator :=(Tvector3_extended): Tvector3_single

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector3_single
                                                    (const v: Tvector3_extended)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

18.3.272 operator :=(Tvector3_extended): Tvector4_double

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector4_double
                                                    (const v: Tvector3_extended)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

18.3.273 operator :=(Tvector3_extended): Tvector4_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector4_extended
                                                    (const v: Tvector3_extende
                                                     : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

18.3.274 operator :=(Tvector3_extended): Tvector4_single

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector3_extended): Tvector4_single
                                                  (const v: Tvector3_extended)
                                                   : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

18.3.275 operator :=(Tvector3_single): Tvector2_double

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector2_double
                                                 (const v: Tvector3_single)
                                                  : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

18.3.276 operator :=(Tvector3_single): Tvector2_extended

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector2_extended
                                                  (const v: Tvector3_single)
                                                   : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

18.3.277 operator :=(Tvector3_single): Tvector2_single

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional single precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector2_single
                                                    (const v: Tvector3_single)
                                                     : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away.

18.3.278 operator :=(Tvector3_single): Tvector3_double

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector3_double
                                                    (const v: Tvector3_single)
                                                     : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected.

18.3.279 operator :=(Tvector3_single): Tvector3_extended

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector3_extended
                                                      (const v: Tvector3_single)
                                                       : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected.

18.3.280 operator :=(Tvector3_single): Tvector4_double

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector4_double
                                                    (const v: Tvector3_single)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

18.3.281 operator :=(Tvector3_single): Tvector4_extended

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector4_extended
                                                    (const v: Tvector3_single)
                                                     : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

18.3.282 operator :=(Tvector3_single): Tvector4_single

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector3_single): Tvector4_single
                                                  (const v: Tvector3_single)
                                                   : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0.

18.3.283 operator :=(Tvector4_double): Tvector2_double

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector2_double
                                                  (const v: Tvector4_double)
                                                   : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

18.3.284 operator :=(Tvector4_double): Tvector2_extended

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector2_extended
                                                    (const v: Tvector4_double)
                                                     : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

18.3.285 operator :=(Tvector4_double): Tvector2_single

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector2_single
                                                    (const v: Tvector4_double)
                                                     : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

18.3.286 operator :=(Tvector4_double): Tvector3_double

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector3_double
                                                    (const v: Tvector4_double)
                                                     : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

18.3.287 operator :=(Tvector4_double): Tvector3_extended

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector3_extended
                                                      (const v: Tvector4_double)
                                                       : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

18.3.288 operator :=(Tvector4_double): Tvector3_single

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector3_single
                                                    (const v: Tvector4_double)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

18.3.289 operator :=(Tvector4_double): Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector4_extended
                                                      (const v: Tvector4_double)
                                                       : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected.

18.3.290 operator :=(Tvector4_double): Tvector4_single

Synopsis: Allow assignment of four-dimensional double precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_double): Tvector4_single
                                                    (const v: Tvector4_double)
                                                     : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

18.3.291 operator :=(Tvector4_extended): Tvector2_double

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector2_double
                                                      (const v: Tvector4_extended)
                                                       : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

18.3.292 operator :=(Tvector4_extended): Tvector2_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector2_extended
                                                    (const v: Tvector4_extende
                                                     : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

18.3.293 operator :=(Tvector4_extended): Tvector2_single

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector2_single
                                                  (const v: Tvector4_extended)
                                                   : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is los because of the conversion.

18.3.294 operator :=(Tvector4_extended): Tvector3_double

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector3_double
                                                  (const v: Tvector4_extended)
                                                   : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

18.3.295 operator :=(Tvector4_extended): Tvector3_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector3_extended
                                                    (const v: Tvector4_extende
                                                     : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimensions are thrown away.

18.3.296 operator :=(Tvector4_extended): Tvector3_single

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector3_single
                                                    (const v: Tvector4_extended)
                                                     : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is los because of the conversion.

18.3.297 operator :=(Tvector4_extended): Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector4_double
                                                    (const v: Tvector4_extended)
                                                     : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion.

18.3.298 operator :=(Tvector4_extended): Tvector4_single

Synopsis: Allow assignment of four-dimensional extended precision vector to four-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_extended): Tvector4_single
                                                    (const v: Tvector4_extended)
                                                     : Tvector4_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

18.3.299 operator :=(Tvector4_single): Tvector2_double

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector2_double
                                                    (const v: Tvector4_single)
                                                     : Tvector2_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

18.3.300 operator :=(Tvector4_single): Tvector2_extended

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector2_extended
                                                      (const v: Tvector4_single)
                                                       : Tvector2_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

18.3.301 operator :=(Tvector4_single): Tvector2_single

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector2_single
                                                    (const v: Tvector4_single)
                                                     : Tvector2_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away.

18.3.302 operator :=(Tvector4_single): Tvector3_double

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector3_double
                                                    (const v: Tvector4_single)
                                                     : Tvector3_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

18.3.303 operator :=(Tvector4_single): Tvector3_extended

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector3_extended
                                                    (const v: Tvector4_single)
                                                     : Tvector3_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

18.3.304 operator :=(Tvector4_single): Tvector3_single

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional single precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector3_single
                                                  (const v: Tvector4_single)
                                                   : Tvector3_single
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away.

18.3.305 operator :=(Tvector4_single): Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector4_double
                                                  (const v: Tvector4_single)
                                                   : Tvector4_double
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected.

18.3.306 operator :=(Tvector4_single): Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector

Declaration:
```
function operator :=(Tvector4_single): Tvector4_extended
                                                    (const v: Tvector4_single)
                                                     : Tvector4_extended
```

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected.

18.3.307 operator ><(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration:
```
function operator ><(Tvector3_double, Tvector3_double): Tvector3_double
                                                                  (const x: Tve
                                                                  const y: Tvec
                                                                   : Tvector3_d
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

18.3.308 operator ><(Tvector3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration:
```
                                                                  functio
                                                                  (const
                                                                  const y
                                                                   : Tvec
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

18.3.309 operator ><(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration:
```
function operator ><(Tvector3_single, Tvector3_single): Tvector3_single
                                                                  (const x: Tve
                                                                  const y: Tvec
                                                                   : Tvector3_s
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

18.4 Tmatrix2_double

18.4.1 Description

The `Tmatrix2_double` object provides a matrix of 2*2 double precision scalars.

18.4.2 Method overview

Page	Property	Description
811	determinant	Calculates the determinant of the matrix.
811	get_column	Returns the c-th column of the matrix as vector.
811	get_row	Returns the r-th row of the matrix as vector.
810	init	Initializes the matrix, setting its elements to the values passed to the constructor.
810	init_identity	Initializes the matrix and sets its elements to the identity matrix.
810	init_zero	Initializes the matrix and sets its elements to zero
811	inverse	Calculates the inverse of the matrix.
811	set_column	Sets c-th column of the matrix with a vector.
811	set_row	Sets r-th row of the matrix with a vector.
812	transpose	Returns the transposition of the matrix.

18.4.3 Tmatrix2_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.4.4 Tmatrix2_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.4.5 Tmatrix2_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: double;ab: double;ba: double;bb: double)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.4.6 Tmatrix2_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_double`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.4.7 Tmatrix2_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_double`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.4.8 Tmatrix2_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector2_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.4.9 Tmatrix2_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector2_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.4.10 Tmatrix2_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : double`

Visibility: default

Description: Returns the determinant of the matrix.

18.4.11 Tmatrix2_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: double) : Tmatrix2_double`

Visibility: default

Description: Tmatrix2_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

18.4.12 Tmatrix2_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.5 Tmatrix2_extended

18.5.1 Description

The `Tmatrix2_extended` object provides a matrix of 2*2 extended precision scalars.

18.5.2 Method overview

Page	Property	Description
813	determinant	Calculates the determinant of the matrix.
813	get_column	Returns the c-th column of the matrix as vector.
813	get_row	Returns the r-th row of the matrix as vector.
813	init	Initializes the matrix, setting its elements to the values passed to the constructor.
812	init_identity	Initializes the matrix and sets its elements to the identity matrix.
812	init_zero	Initializes the matrix and sets its elements to zero
814	inverse	Calculates the inverse of the matrix.
813	set_column	Sets c-th column of the matrix with a vector.
813	set_row	Sets r-th row of the matrix with a vector.
814	transpose	Returns the transposition of the matrix.

18.5.3 Tmatrix2_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.5.4 Tmatrix2_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.5.5 Tmatrix2_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: extended;ab: extended;ba: extended;bb: extended)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.5.6 Tmatrix2_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.5.7 Tmatrix2_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.5.8 Tmatrix2_extended.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector2_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.5.9 Tmatrix2_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector2_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.5.10 Tmatrix2_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

18.5.11 Tmatrix2_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix2_extended`

Visibility: default

Description: Tmatrix2_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

18.5.12 Tmatrix2_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.6 Tmatrix2_single

18.6.1 Description

The `Tmatrix2_single` object provides a matrix of 2*2 single precision scalars.

18.6.2 Method overview

Page	Property	Description
816	determinant	Calculates the determinant of the matrix.
815	get_column	Returns the c-th column of the matrix as vector.
815	get_row	Returns the r-th row of the matrix as vector.
815	init	Initializes the matrix, setting its elements to the values passed to the constructor.
815	init_identity	Initializes the matrix and sets its elements to the identity matrix.
814	init_zero	Initializes the matrix and sets its elements to zero
816	inverse	Calculates the inverse of the matrix.
815	set_column	Sets c-th column of the matrix with a vector.
816	set_row	Sets r-th row of the matrix with a vector.
816	transpose	Returns the transposition of the matrix.

18.6.3 Tmatrix2_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.6.4 Tmatrix2_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.6.5 Tmatrix2_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: single;ab: single;ba: single;bb: single)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.6.6 Tmatrix2_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_single`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.6.7 Tmatrix2_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_single`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.6.8 Tmatrix2_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector2_single)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.6.9 Tmatrix2_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector2_single)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.6.10 Tmatrix2_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

18.6.11 Tmatrix2_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix2_single`

Visibility: default

Description: Tmatrix2_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

18.6.12 Tmatrix2_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.7 Tmatrix3_double

18.7.1 Description

The `Tmatrix3_double` object provides a matrix of 3*3 double precision scalars.

18.7.2 Method overview

Page	Property	Description
818	determinant	Calculates the determinant of the matrix.
817	get_column	Returns the c-th column of the matrix as vector.
818	get_row	Returns the r-th row of the matrix as vector.
817	init	Initializes the matrix, setting its elements to the values passed to the constructor.
817	init_identity	Initializes the matrix and sets its elements to the identity matrix.
817	init_zero	Initializes the matrix and sets its elements to zero
818	inverse	Calculates the inverse of the matrix.
818	set_column	Sets c-th column of the matrix with a vector.
818	set_row	Sets r-th row of the matrix with a vector.
818	transpose	Returns the transposition of the matrix.

18.7.3 Tmatrix3_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.7.4 Tmatrix3_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.7.5 Tmatrix3_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: double;ab: double;ac: double;ba: double;bb: double;
                 bc: double;ca: double;cb: double;cc: double)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.7.6 Tmatrix3_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_double`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.7.7 Tmatrix3_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_double`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.7.8 Tmatrix3_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector3_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.7.9 Tmatrix3_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector3_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.7.10 Tmatrix3_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : double`

Visibility: default

Description: Returns the determinant of the matrix.

18.7.11 Tmatrix3_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: double) : Tmatrix3_double`

Visibility: default

Description: Tmatrix3_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

18.7.12 Tmatrix3_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.8 Tmatrix3_extended

18.8.1 Description

The `Tmatrix3_extended` object provides a matrix of 3*3 extended precision scalars.

18.8.2 Method overview

Page	Property	Description
820	determinant	Calculates the determinant of the matrix.
820	get_column	Returns the c-th column of the matrix as vector.
820	get_row	Returns the r-th row of the matrix as vector.
819	init	Initializes the matrix, setting its elements to the values passed to the constructor.
819	init_identity	Initializes the matrix and sets its elements to the identity matrix.
819	init_zero	Initializes the matrix and sets its elements to zero
820	inverse	Calculates the inverse of the matrix.
820	set_column	Sets r-th column of the matrix with a vector.
820	set_row	Sets r-th row of the matrix with a vector.
821	transpose	Returns the transposition of the matrix.

18.8.3 Tmatrix3_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration:
```
constructor init_zero
```

Visibility: default

18.8.4 Tmatrix3_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.8.5 Tmatrix3_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: extended;ab: extended;ac: extended;ba: extended;
                 bb: extended;bc: extended;ca: extended;cb: extended;
                 cc: extended)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.8.6 Tmatrix3_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.8.7 Tmatrix3_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.8.8 Tmatrix3_extended.set_column

Synopsis: Sets r-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector3_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.8.9 Tmatrix3_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector3_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.8.10 Tmatrix3_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

18.8.11 Tmatrix3_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix3_extended`

Visibility: default

Description: Tmatrix3_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

18.8.12 Tmatrix3_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.9 Tmatrix3_single

18.9.1 Description

The `Tmatrix3_single` object provides a matrix of 3*3 single precision scalars.

18.9.2 Method overview

Page	Property	Description
823	determinant	Calculates the determinant of the matrix.
822	get_column	Returns the c-th column of the matrix as vector.
822	get_row	Returns the r-th row of the matrix as vector.
822	init	Initializes the matrix, setting its elements to the values passed to the constructor.
821	init_identity	Initializes the matrix and sets its elements to the identity matrix.
821	init_zero	Initializes the matrix and sets its elements to zero
823	inverse	Calculates the inverse of the matrix.
822	set_column	Sets c-th column of the matrix with a vector.
822	set_row	Sets r-th row of the matrix with a vector.
823	transpose	Returns the transposition of the matrix.

18.9.3 Tmatrix3_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.9.4 Tmatrix3_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.9.5 Tmatrix3_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: single;ab: single;ac: single;ba: single;bb: single;
                 bc: single;ca: single;cb: single;cc: single)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.9.6 Tmatrix3_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector3_single
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.9.7 Tmatrix3_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector3_single
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.9.8 Tmatrix3_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration:
```
procedure set_column(c: Byte;const v: Tvector3_single)
```

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.9.9 Tmatrix3_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration:
```
procedure set_row(r: Byte;const v: Tvector3_single)
```

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.9.10 Tmatrix3_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

18.9.11 Tmatrix3_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix3_single`

Visibility: default

Description: Tmatrix3_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

18.9.12 Tmatrix3_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.10 Tmatrix4_double

18.10.1 Description

The `Tmatrix4_double` object provides a matrix of 4*4 double precision scalars.

18.10.2 Method overview

Page	Property	Description
825	determinant	Calculates the determinant of the matrix.
824	get_column	Returns the c-th column of the matrix as vector.
824	get_row	Returns the r-th row of the matrix as vector.
824	init	Initializes the matrix, setting its elements to the values passed to the constructor.
824	init_identity	Initializes the matrix and sets its elements to the identity matrix.
824	init_zero	Initializes the matrix and sets its elements to zero
825	inverse	Calculates the inverse of the matrix.
825	set_column	Sets c-th column of the matrix with a vector.
825	set_row	Sets r-th row of the matrix with a vector.
825	transpose	Returns the transposition of the matrix.

18.10.3 Tmatrix4_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration:
```
constructor init_zero
```

Visibility: default

18.10.4 Tmatrix4_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.10.5 Tmatrix4_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: double;ab: double;ac: double;ad: double;ba: double;
                 bb: double;bc: double;bd: double;ca: double;cb: double;
                 cc: double;cd: double;da: double;db: double;dc: double;
                 dd: double)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.10.6 Tmatrix4_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector4_double
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.10.7 Tmatrix4_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector4_double
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.10.8 Tmatrix4_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector4_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.10.9 Tmatrix4_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector4_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.10.10 Tmatrix4_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : double`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

18.10.11 Tmatrix4_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: double) : Tmatrix4_double`

Visibility: default

Description: Tmatrix4_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

18.10.12 Tmatrix4_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.11 Tmatrix4_extended

18.11.1 Description

The `Tmatrix4_extended` object provides a matrix of 4*4 extended precision scalars.

18.11.2 Method overview

Page	Property	Description
827	determinant	Calculates the determinant of the matrix.
827	get_column	Returns the c-th column of the matrix as vector.
827	get_row	Returns the r-th row of the matrix as vector.
826	init	Initializes the matrix, setting its elements to the values passed to the constructor.
826	init_identity	Initializes the matrix and sets its elements to the identity matrix.
826	init_zero	Initializes the matrix and sets its elements to zero
828	inverse	Calculates the inverse of the matrix.
827	set_column	Sets c-th column of the matrix with a vector.
827	set_row	Sets r-th row of the matrix with a vector.
828	transpose	Returns the transposition of the matrix.

18.11.3 Tmatrix4_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration:
```
constructor init_zero
```

Visibility: default

18.11.4 Tmatrix4_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.11.5 Tmatrix4_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: extended;ab: extended;ac: extended;ad: extended;
                 ba: extended;bb: extended;bc: extended;bd: extended;
                 ca: extended;cb: extended;cc: extended;cd: extended;
                 da: extended;db: extended;dc: extended;dd: extended)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.11.6 Tmatrix4_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.11.7 Tmatrix4_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.11.8 Tmatrix4_extended.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector4_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.11.9 Tmatrix4_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector4_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.11.10 Tmatrix4_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

18.11.11 Tmatrix4_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix4_extended`

Visibility: default

Description: Tmatrix4_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

18.11.12 Tmatrix4_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.12 Tmatrix4_single

18.12.1 Description

The `Tmatrix4_single` object provides a matrix of 4*4 single precision scalars.

18.12.2 Method overview

Page	Property	Description
830	determinant	Calculates the determinant of the matrix.
829	get_column	Returns the c-th column of the matrix as vector.
829	get_row	Returns the r-th row of the matrix as vector.
829	init	Initializes the matrix, setting its elements to the values passed to the constructor.
829	init_identity	Initializes the matrix and sets its elements to the identity matrix.
828	init_zero	Initializes the matrix and sets its elements to zero
830	inverse	Calculates the inverse of the matrix.
829	set_column	Sets c-th column of the matrix with a vector.
830	set_row	Sets r-th row of the matrix with a vector.
830	transpose	Returns the transposition of the matrix.

18.12.3 Tmatrix4_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.12.4 Tmatrix4_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

18.12.5 Tmatrix4_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: single;ab: single;ac: single;ad: single;ba: single;
                 bb: single;bc: single;bd: single;ca: single;cb: single;
                 cc: single;cd: single;da: single;db: single;dc: single;
                 dd: single)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

18.12.6 Tmatrix4_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector4_single
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

18.12.7 Tmatrix4_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector4_single
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

18.12.8 Tmatrix4_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration:
```
procedure set_column(c: Byte;const v: Tvector4_single)
```

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

18.12.9 Tmatrix4_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector4_single)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

18.12.10 Tmatrix4_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

18.12.11 Tmatrix4_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix4_single`

Visibility: default

Description: Tmatrix4_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

18.12.12 Tmatrix4_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

18.13 Tvector2_double

18.13.1 Description

The `Tvector2_double` object provides a vector of two double precision scalars.

18.13.2 Method overview

Page	Property	Description
831	init	Initializes the vector, setting its elements to the values passed to the constructor.
831	init_one	Initializes the vector and sets its elements to one
831	init_zero	Initializes the vector and sets its elements to zero
831	length	Calculates the length of the vector.
831	squared_length	Calculates the squared length of the vector.

18.13.3 Tvector2_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.13.4 Tvector2_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.13.5 Tvector2_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: double;b: double)`

Visibility: default

18.13.6 Tvector2_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (831) if you are able to, as it is faster.

18.13.7 Tvector2_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

18.14 Tvector2_extended

18.14.1 Description

The `Tvector2_extended` object provides a vector of two extended precision scalars.

18.14.2 Method overview

Page	Property	Description
832	init	Initializes the vector, setting its elements to the values passed to the constructor.
832	init_one	Initializes the vector and sets its elements to one
832	init_zero	Initializes the vector and sets its elements to zero
832	length	Calculates the length of the vector.
833	squared_length	Calculates the squared length of the vector.

18.14.3 Tvector2_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.14.4 Tvector2_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.14.5 Tvector2_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended;b: extended)`

Visibility: default

18.14.6 Tvector2_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (833) if you are able to, as it is faster.

18.14.7 Tvector2_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

18.15 Tvector2_single

18.15.1 Description

The `Tvector2_single` object provides a vector of two single precision scalars.

18.15.2 Method overview

Page	Property	Description
833	init	Initializes the vector, setting its elements to the values passed to the constructor.
833	init_one	Initializes the vector and sets its elements to one
833	init_zero	Initializes the vector and sets its elements to zero
834	length	Calculates the length of the vector.
834	squared_length	Calculates the squared length of the vector.

18.15.3 Tvector2_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.15.4 Tvector2_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.15.5 Tvector2_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single;b: single)`

Visibility: default

18.15.6 Tvector2_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (834) if you are able to, as it is faster.

18.15.7 Tvector2_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

18.16 Tvector3_double

18.16.1 Description

The `Tvector3_double` object provides a vector of three double precision scalars.

18.16.2 Method overview

Page	Property	Description
835	init	Initializes the vector, setting its elements to the values passed to the constructor.
834	init_one	Initializes the vector and sets its elements to one
834	init_zero	Initializes the vector and sets its elements to zero
835	length	Calculates the length of the vector.
835	squared_length	Calculates the squared length of the vector.

18.16.3 Tvector3_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.16.4 Tvector3_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.16.5 Tvector3_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: double;b: double;c: double)`

Visibility: default

18.16.6 Tvector3_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (835) if you are able to, as it is faster.

18.16.7 Tvector3_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

18.17 Tvector3_extended

18.17.1 Description

The `Tvector3_extended` object provides a vector of three extended precision scalars.

18.17.2 Method overview

Page	Property	Description
836	init	Initializes the vector, setting its elements to the values passed to the constructor.
836	init_one	Initializes the vector and sets its elements to one
835	init_zero	Initializes the vector and sets its elements to zero
836	length	Calculates the length of the vector.
836	squared_length	Calculates the squared length of the vector.

18.17.3 Tvector3_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.17.4 Tvector3_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.17.5 Tvector3_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended;b: extended;c: extended)`

Visibility: default

18.17.6 Tvector3_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (836) if you are able to, as it is faster.

18.17.7 Tvector3_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

18.18 Tvector3_single

18.18.1 Description

The `Tvector3_single` object provides a vector of three single precision scalars.

18.18.2 Method overview

Page	Property	Description
837	init	Initializes the vector, setting its elements to the values passed to the constructor.
837	init_one	Initializes the vector and sets its elements to one
837	init_zero	Initializes the vector and sets its elements to zero
837	length	Calculates the length of the vector.
837	squared_length	Calculates the squared length of the vector.

18.18.3 Tvector3_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.18.4 Tvector3_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.18.5 Tvector3_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single;b: single;c: single)`

Visibility: default

18.18.6 Tvector3_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (837) if you are able to, as it is faster.

18.18.7 Tvector3_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

18.19 Tvector4_double

18.19.1 Description

The `Tvector4_double` object provides a vector of four double precision scalars.

18.19.2 Method overview

Page	Property	Description
838	init	Initializes the vector, setting its elements to the values passed to the constructor.
838	init_one	Initializes the vector and sets its elements to one
838	init_zero	Initializes the vector and sets its elements to zero
838	length	Calculates the length of the vector.
838	squared_length	Calculates the squared length of the vector.

18.19.3 Tvector4_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.19.4 Tvector4_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.19.5 Tvector4_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: double;b: double;c: double;d: double)`

Visibility: default

18.19.6 Tvector4_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (838) if you are able to, as it is faster.

18.19.7 Tvector4_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

18.20 Tvector4_extended

18.20.1 Description

The `Tvector4_extended` object provides a vector of four extended precision scalars.

18.20.2 Method overview

Page	Property	Description
839	init	Initializes the vector, setting its elements to the values passed to the constructor.
839	init_one	Initializes the vector and sets its elements to one
839	init_zero	Initializes the vector and sets its elements to zero
839	length	Calculates the length of the vector.
840	squared_length	Calculates the squared length of the vector.

18.20.3 Tvector4_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.20.4 Tvector4_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.20.5 Tvector4_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended;b: extended;c: extended;d: extended)`

Visibility: default

18.20.6 Tvector4_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (840) if you are able to, as it is faster.

18.20.7 Tvector4_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

18.21 Tvector4_single

18.21.1 Description

The `Tvector4_single` object provides a vector of four single precision scalars.

18.21.2 Method overview

Page	Property	Description
840	init	Initializes the vector, setting its elements to the values passed to the constructor.
840	init_one	Initializes the vector and sets its elements to one
840	init_zero	Initializes the vector and sets its elements to zero
841	length	Calculates the length of the vector.
841	squared_length	Calculates the squared length of the vector.

18.21.3 Tvector4_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

18.21.4 Tvector4_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

18.21.5 Tvector4_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single;b: single;c: single;d: single)`

Visibility: default

18.21.6 Tvector4_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (841) if you are able to, as it is faster.

18.21.7 Tvector4_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

Chapter 19

Reference for unit 'mmx'

19.1 Overview

This document describes the MMX unit. This unit allows you to use the MMX capabilities of the Free Pascal compiler. It was written by Florian Klaempfl for the I386 processor. It should work on all platforms that use the Intel processor.

19.2 Constants, types and variables

19.2.1 Constants

```
is_amd_3d_cpu : Boolean = false
```

The is_amd_3d_cpu initialized constant allows you to determine if the computer has the AMD 3D extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_dsp_cpu : Boolean = false
```

The is_amd_3d_dsp_cpu initialized constant allows you to determine if the computer has the AMD 3D DSP extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_mmx_cpu : Boolean = false
```

The is_amd_3d_mmx_cpu initialized constant allows you to determine if the computer has the AMD 3D MMX extensions. It is set correctly in the unit's initialization code.

```
is_mmx_cpu : Boolean = false
```

The is_mmx_cpu initialized constant allows you to determine if the computer has MMX extensions. It is set correctly in the unit's initialization code.

```
is_sse2_cpu : Boolean = false
```

The is_sse2_cpu initialized constant allows you to determine if the computer has the SSE2 extensions. It is set correctly in the unit's initialization code.

```
is_sse_cpu : Boolean = false
```

The is_sse_cpu initialized constant allows you to determine if the computer has the SSE extensions. It is set correctly in the unit's initialization code.

19.2.2 Types

```
pmmxbyte = ^tmmxbyte
```

Pointer to tmmxbyte (843) array type

```
pmmxcardinal = ^tmmxcardinal
```

Pointer to tmmxcardinal (843) array type

```
pmmxinteger = ^tmmxinteger
```

Pointer to tmmxinteger (843) array type

```
pmmxlongint = ^tmmxlongint
```

Pointer to tmmxlongint (843) array type

```
pmmxshortint = ^tmmxshortint
```

Pointer to tmmxshortint (843) array type

```
pmmxsingle = ^tmmxsingle
```

Pointer to tmmxsingle (843) array type

```
pmmxword = ^tmmxword
```

Pointer to tmmxword (843) array type

```
tmmxbyte = Array[0..7] of Byte
```

Array of bytes, 64 bits in size

```
tmmxcardinal = Array[0..1] of cardinal
```

Array of cardinals, 64 bits in size

```
tmmxinteger = Array[0..3] of Integer
```

Array of integers, 64 bits in size

```
tmmxlongint = Array[0..1] of LongInt
```

Array of longint, 64 bits in size

```
tmmxshortint = Array[0..7] of ShortInt
```

Array of shortints, 64 bits in size

```
tmmxsingle = Array[0..1] of single
```

Array of singles, 64 bits in size

```
tmmxword = Array[0..3] of Word
```

Array of words, 64 bits in size

19.3 Procedures and functions

19.3.1 emms

Synopsis: Reset floating point registers

Declaration: `procedure emms`

Visibility: default

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. The following code demonstrates this:

```
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.
```

See also: femms (844)

19.3.2 femms

Synopsis: Reset floating point registers - AMD version

Declaration: `procedure femms`

Visibility: default

Description: `femms` executes the `femms` assembler instruction for AMD processors. it is not supported by all assemblers, hence it is coded as byte codes.

See also: emms (844)

Chapter 20

Reference for unit 'Mouse'

20.1 Writing a custom mouse driver

The mouse unit has support for adding a custom mouse driver. This can be used to add support for mouses not supported by the standard Free Pascal driver, but also to enhance an existing driver for instance to log mouse events or to implement a record and playback function.

The following unit shows how a mouse driver can be enhanced by adding some logging capabilities to the driver.

20.2 Overview

The Mouse unit implements a platform independent mouse handling interface. It is implemented identically on all platforms supported by Free Pascal and can be enhanced with custom drivers, should this be needed. It is intended to be used only in text-based screens, for instance in conjunction with the keyboard and video unit. No support for graphical screens is implemented, and there are (currently) no plans to implement this.

20.3 Constants, types and variables

20.3.1 Constants

```
errMouseBase = 1030
```

Base for mouse error codes.

```
errMouseInitError = errMouseBase + 0
```

Mouse initialization error

```
errMouseNotImplemented = errMouseBase + 1
```

Mouse driver not implemented.

```
MouseActionDown = $0001
```

Mouse button down event signal.

```
MouseActionMove = $0004
```

Mouse cursor move event signal.

```
MouseActionUp = $0002
```

Mouse button up event signal.

```
MouseEventBufSize = 16
```

The mouse unit has a mechanism to buffer mouse events. This constant defines the size of the event buffer.

```
MouseLeftButton = $01
```

Left mouse button event.

```
MouseMiddleButton = $04
```

Middle mouse button event.

```
MouseRightButton = $02
```

Right mouse button event.

20.3.2 Types

```
PMouseEvent = ^TMouseEvent
```

Pointer to TMouseEvent (847) record.

```
TMouseDriver = record
  UseDefaultQueue : Boolean;
  InitDriver : procedure;
  DoneDriver : procedure;
  DetectMouse : function : Byte;
  ShowMouse : procedure;
  HideMouse : procedure;
  GetMouseX : function : Word;
  GetMouseY : function : Word;
  GetMouseButtons : function : Word;
  SetMouseXY : procedure(x: Word;y: Word);
  GetMouseEvent : procedure(var MouseEvent: TMouseEvent);
  PollMouseEvent : function(var MouseEvent: TMouseEvent) : Boolean;
  PutMouseEvent : procedure(const MouseEvent: TMouseEvent);
end
```

The `TMouseDriver` record is used to implement a mouse driver in the SetMouseDriver (852) function. Its fields must be filled in before calling the SetMouseDriver (852) function.

```
TMouseEvent = packed record
  buttons : Word;
  x : Word;
  y : Word;
  Action : Word;
end
```

The `TMouseEvent` is the central type of the mouse unit, it is used to describe all mouse events.

The `Buttons` field describes which buttons were down when the event occurred. The `x,y` fields describe where the event occurred on the screen. The `Action` describes what action was going on when the event occurred. The `Buttons` and `Action` field can be examined using the constants defined in the unit interface.

20.3.3 Variables

`MouseButtons : Byte`

This variable keeps track of the last known mouse button state. Do not use.

`MouseIntFlag : Byte`

This variable keeps track of the last known internal mouse state. Do not use.

`MouseWhereX : Word`

This variable keeps track of the last known cursor position. Do not use.

`MouseWhereY : Word`

This variable keeps track of the last known cursor position. Do not use.

20.4 Procedures and functions

20.4.1 DetectMouse

Synopsis: Detect the presence of a mouse.

Declaration: `function DetectMouse : Byte`

Visibility: default

Description: `DetectMouse` detects whether a mouse is attached to the system or not. If there is no mouse, then zero is returned. If a mouse is attached, then the number of mouse buttons is returnead.

This function should be called after the mouse driver was initialized.

Errors: None.

See also: InitMouse (851), DoneMouse (848)

Listing: ./mouseex/ex1.pp

```
Program Example1;

{ Program to demonstrate the DetectMouse function. }

Uses mouse;

Var
  Buttons : Byte;

begin
  InitMouse;
  Buttons:=DetectMouse;
  If Buttons=0 then
    WriteIn('No mouse present.')
  else
    WriteIn('Found mouse with ',Buttons,' buttons.');
  DoneMouse;
end.
```

20.4.2 DoneMouse

Synopsis: Deinitialize mouse driver.

Declaration: `procedure DoneMouse`

Visibility: default

Description: `DoneMouse` De-initializes the mouse driver. It cleans up any memory allocated when the mouse was initialized, or removes possible mouse hooks from memory. The mouse functions will not work after `DoneMouse` was called. If `DoneMouse` is called a second time, it will exit at once. `InitMouse` should be called before `DoneMouse` can be called again.

For an example, see most other mouse functions.

Errors: None.

See also: DetectMouse (847), InitMouse (851)

20.4.3 GetMouseButtons

Synopsis: Get the state of the mouse buttons

Declaration: `function GetMouseButtons : Word`

Visibility: default

Description: `GetMouseButtons` returns the current button state of the mouse, i.e. it returns a or-ed combination of the following constants:

MouseLeftButtonWhen the left mouse button is held down.

MouseRightButtonWhen the right mouse button is held down.

MouseMiddleButtonWhen the middle mouse button is held down.

Errors: None.

See also: GetMouseEvent (849), GetMouseX (849), GetMouseY (850)

Listing: ./mouseex/ex2.pp

```
Program Example2;

{ Program to demonstrate the GetMouseButtons function. }

Uses mouse;

begin
  InitMouse;
  Writeln('Press right mouse button to exit program');
  While (GetMouseButtons<>MouseRightButton) do ;
  DoneMouse;
end.
```

20.4.4 GetMouseDriver

Synopsis: Get a copy of the currently active mouse driver.

Declaration: `procedure GetMouseDriver(var Driver: TMouseDriver)`

Visibility: default

Description: `GetMouseDriver` returns the currently set mouse driver. It can be used to retrieve the current mouse driver, and override certain callbacks.

A more detailed explanation about getting and setting mouse drivers can be found in mousedrv (845).

For an example, see the section on writing a custom mouse driver, mousedrv (845)

Errors: None.

See also: SetMouseDriver (852)

20.4.5 GetMouseEvent

Synopsis: Get next mouse event from the queue.

Declaration: `procedure GetMouseEvent(var MouseEvent: TMouseEvent)`

Visibility: default

Description: `GetMouseEvent` returns the next mouse event (a movement, button press or button release), and waits for one if none is available in the queue.

Some mouse drivers can implement a mouse event queue which can hold multiple events till they are fetched. Others don't, and in that case, a one-event queue is implemented for use with PollMouseEvent (852).

Errors: None.

See also: GetMouseButtons (848), GetMouseX (849), GetMouseY (850)

20.4.6 GetMouseX

Synopsis: Query the current horizontal position of the mouse cursor.

Declaration: `function GetMouseX : Word`

Visibility: default

Description: `GetMouseX` returns the current `X` position of the mouse. `X` is measured in characters, starting at 0 for the left side of the screen.

Errors: None.

See also: GetMouseButtons (848), GetMouseEvent (849), GetMouseY (850)

Listing: ./mouseex/ex4.pp

```
Program Example4;

{ Program to demonstrate the GetMouseX,GetMouseY functions. }

Uses mouse;

Var
  X,Y : Word;

begin
  InitMouse;
  Writeln('Move mouse cursor to square 10,10 to end');
  Repeat
    X:=GetMouseX;
    Y:=GetMouseY;
    Writeln('X,Y= (',X,',',Y,')');
  Until (X=9) and (Y=9);
  DoneMouse;
end.
```

20.4.7 GetMouseY

Synopsis: Query the current vertical position of the mouse cursor.

Declaration: `function GetMouseY : Word`

Visibility: default

Description: `GetMouseY` returns the current `Y` position of the mouse. `Y` is measured in characters, starting at 0 for the top of the screen.

For an example, see GetMouseX (849)

Errors: None.

See also: GetMouseButtons (848), GetMouseEvent (849), GetMouseX (849)

20.4.8 HideMouse

Synopsis: Hide the mouse cursor.

Declaration: `procedure HideMouse`

Visibility: default

Description: `HideMouse` hides the mouse cursor. This may or may not be implemented on all systems, and depends on the driver.

Errors: None.

See also: ShowMouse (853)

Listing: ./mouseex/ex5.pp

```
Program Example5;

{ Program to demonstrate the HideMouse function. }

Uses mouse;

Var
  Event : TMouseEvent;
  Visible: Boolean;

begin
  InitMouse;
  ShowMouse;
  Visible:=True;
  Writeln('Press left mouse button to hide/show, right button quits');
  Repeat
   GetMouseEvent(Event);
   With Event do
     If (Buttons=MouseLeftbutton) and
        (Action=MouseActionDown) then
       begin
       If Visible then
         HideMouse
       else
         ShowMouse;
       Visible:=Not Visible;
       end;
  Until (Event.Buttons=MouseRightButton) and
        (Event.Action=MouseActionDown);
  DoneMouse;
end.
```

20.4.9 InitMouse

Synopsis: Initialize the FPC mouse driver.

Declaration: `procedure InitMouse`

Visibility: default

Description: `InitMouse` Initializes the mouse driver. This will allocate any data structures needed for the mouse to function. All mouse functions can be used after a call to `InitMouse`.

A call to `InitMouse` must always be followed by a call to DoneMouse (848) at program exit. Failing to do so may leave the mouse in an unusable state, or may result in memory leaks.

For an example, see most other functions.

Errors: None.

See also: DoneMouse (848), DetectMouse (847)

20.4.10 PollMouseEvent

Synopsis: Query next mouse event. Do not wait if none available.

Declaration: `function PollMouseEvent(var MouseEvent: TMouseEvent) : Boolean`

Visibility: default

Description: `PollMouseEvent` checks whether a mouse event is available, and returns it in `MouseEvent` if one is found. The function result is `True` in that case. If no mouse event is pending, the function result is `False`, and the contents of `MouseEvent` is undefined.

Note that after a call to `PollMouseEvent`, the event should still be removed from the mouse event queue with a call to `GetMouseEvent`.

Errors: None.

See also: GetMouseEvent (849), PutMouseEvent (852)

20.4.11 PutMouseEvent

Synopsis: Put a mouse event in the venet queue.

Declaration: `procedure PutMouseEvent(const MouseEvent: TMouseEvent)`

Visibility: default

Description: `PutMouseEvent` adds `MouseEvent` to the input queue. The next call to GetMouseEvent (849) or `PollMouseEvent` will then return `MouseEvent`.

Please note that depending on the implementation the mouse event queue can hold only one value.

Errors: None.

See also: GetMouseEvent (849), PollMouseEvent (852)

20.4.12 SetMouseDriver

Synopsis: Set a new mouse driver.

Declaration: `procedure SetMouseDriver(const Driver: TMouseDriver)`

Visibility: default

Description: `SetMouseDriver` sets the mouse driver to `Driver`. This function should be called before InitMouse (851) is called, or after `DoneMouse` is called. If it is called after the mouse has been initialized, it does nothing.

For more information on setting the mouse driver, mousedrv (845).

For an example, see mousedrv (845)

Errors:

See also: InitMouse (851), DoneMouse (848), GetMouseDriver (849)

20.4.13 SetMouseXY

Synopsis: Set the mouse cursor position.

Declaration: `procedure SetMouseXY(x: Word;y: Word)`

Visibility: default

Description: `SetMouseXY` places the mouse cursor on `X,Y`. X and Y are zero based character coordinates: `0,0` is the top-left corner of the screen, and the position is in character cells (i.e. not in pixels).

Errors: None.

See also: GetMouseX (849), GetMouseY (850)

Listing: ./mouseex/ex7.pp

```
Program Example7;

{ Program to demonstrate the SetMouseXY function. }

Uses mouse;

begin
  InitMouse;
  Writeln('Click right mouse button to quit.');
  SetMouseXY(40,12);
  Repeat
    Writeln(GetMouseX,',',GetMouseY);
    If (GetMouseX>70) then
      SetMouseXY(10,GetMouseY);
    If (GetMouseY>20) then
      SetMouseXY(GetMouseX,5);
  Until (GetMouseButtons=MouseRightButton);
  DoneMouse;
end.
```

20.4.14 ShowMouse

Synopsis: Show the mouse cursor.

Declaration: `procedure ShowMouse`

Visibility: default

Description: `ShowMouse` shows the mouse cursor if it was previously hidden. The capability to hide or show the mouse cursor depends on the driver.

For an example, see HideMouse (850)

Errors: None.

See also: HideMouse (850)

Chapter 21

Reference for unit 'Objects'

21.1 Overview

This document documents the objects unit. The unit was implemented by many people, and was mainly taken from the FreeVision sources. It has been ported to all supported platforms.

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

21.2 Constants, types and variables

21.2.1 Constants

```
coIndexError = -1
```

Collection list error: Index out of range

```
coOverflow = -2
```

Collection list error: Overflow

```
DefaultTPCompatible : Boolean = false
```

`DefaultTPCompatible` is used to initialize tstream.tpcompatible (**??**).

```
MaxBytes = 128 * 1024 * 128
```

Maximum data size (in bytes)

```
MaxCollectionSize = MaxBytes div SizeOf ( Pointer )
```

Maximum collection size (in items)

```
MaxPtrs = MaxBytes div SizeOf ( Pointer )
```

Maximum data size (in pointers)

```
MaxReadBytes = $7fffffff
```

Maximum data that can be read from a stream (not used)

```
MaxTPCompatibleCollectionSize = 65520 div 4
```

Maximum collection size (in items, same value as in TP)

```
MaxWords = MaxBytes div SizeOf ( Word )
```

Maximum data size (in words)

```
RCollection : TStreamRec = (ObjType:50;VmtLink:Ofs ( TypeOf ( TCollection ) ^ ) ;Loa
```

Default stream record for the TCollection (871) object.

```
RStrCollection : TStreamRec = (ObjType:69;VmtLink:Ofs ( TypeOf ( TStrCollection ) ^
```

Default stream record for the TStrCollection (910) object.

```
RStringCollection : TStreamRec = (ObjType:51;VmtLink:Ofs ( TypeOf ( TStringCollectio
```

Default stream record for the TStringCollection (920) object.

```
RStringList : TStreamRec = (ObjType:52;VmtLink:Ofs ( TypeOf ( TStringList ) ^ ) ;Loa
```

Default stream record for the TStringList (922) object.

```
RStrListMaker : TStreamRec = (ObjType:52;VmtLink:Ofs ( TypeOf ( TStrListMaker ) ^ )
```

Default stream record for the TStrListMaker (924) object.

```
stCreate = $3C00
```

Stream initialization mode: Create new file

```
stError = -1
```

Stream error codes: Access error

```
stGetError = -5
```

Stream error codes: Get object error

```
stInitError = -2
```

Stream error codes: Initialize error

```
stOk = 0
```

Stream error codes: No error

```
stOpen = $3D02
```

Stream initialization mode: Read/write access

```
stOpenError = -8
```

Stream error codes: Error opening stream

```
stOpenRead = $3D00
```

Stream initialization mode: Read access only

```
stOpenWrite = $3D01
```

Stream initialization mode: Write access only

```
stPutError = -6
```

Stream error codes: Put object error

```
stReadError = -3
```

Stream error codes: Stream read error

```
StreamError : Pointer = nil
```

Pointer to default stream error handler.

```
stSeekError = -7
```

Stream error codes: Seek error in stream

```
stWriteError = -4
```

Stream error codes: Stream write error

```
vmtHeaderSize = 8
```

Size of the VMT header in an object (not used).

21.2.2 Types

```
AsciiZ = Array[0..255] of Char
```

Filename - null terminated array of characters.

```
FNameStr = String
```

Filename - shortstring version.

```
LongRec = packed record
  Hi : Word;
  Lo : Word;
end
```

Record describing a longint (in Words)

```
PBufStream = ^TBufStream
```

Pointer to TBufStream (867) object.

```
PByteArray = ^TByteArray
```

Pointer to TByteArray (859)

```
PCharSet = ^TCharSet
```

Pointer to TCharSet (859).

```
PCollection = ^TCollection
```

Pointer to TCollection (871) object.

```
PDosStream = ^TDosStream
```

Pointer to TDosStream (885) object.

```
PItemList = ^TItemList
```

Pointer to TItemList (859) object.

```
PMemoryStream = ^TMemoryStream
```

Pointer to TMemoryStream (890) object.

```
PObject = ^TObject
```

Pointer to TObject (892) object.

```
PPoint = ^TPoint
```

Pointer to TPoint (894) record.

```
PPointerArray = ^TPointerArray
```

Pointer to TPointerArray (859)

```
PRect = ^TRect
```

Pointer to TRect (894) object.

```
PResourceCollection = ^TResourceCollection
```

Pointer to TResourceCollection (900) object.

```
PResourceFile = ^TResourceFile
```

Pointer to TResourceFile (901) object.

```
PSortedCollection = ^TSortedCollection
```

Pointer to TSortedCollection (904) object.

```
PStrCollection = ^TStrCollection
```

Pointer to TStrCollection (910) object.

```
PStream = ^TStream
```

Pointer type to TStream (912)

```
PStreamRec = ^TStreamRec
```

Pointer to TStreamRec (859)

```
PStrIndex = ^TStrIndex
```

Pointer to TStrIndex (859) array.

```
PString = PShortString
```

Pointer to a shortstring.

```
PStringCollection = ^TStringCollection
```

Pointer to TStringCollection (920) object.

```
PStringList = ^TStringList
```

Pointer to TStringList (922) object.

```
PStrListMaker = ^TStrListMaker
```

Pointer to TStrListMaker (924) object.

```
PtrRec = packed record
  Ofs : Word;
  Seg : Word;
end
```

Record describing a pointer to a memory location.

```
PUnSortedStrCollection = ^TUnSortedStrCollection
```

Pointer to TUnsortedStrCollection (925) object.

```
PWordArray = ^TWordArray
```

Pointer to TWordArray (860)

```
Sw_Integer = LongInt
```

Alias for longint

```
Sw_Word = Cardinal
```

Alias for Cardinal

```
TByteArray = Array[0..MaxBytes-1] of Byte
```

Array with maxmimum allowed number of bytes.

```
TCharSet = Set of Char
```

Generic set of characters type.

```
TItemList = Array[0..MaxCollectionSize-1] of Pointer
```

Pointer array type used in a TCollection (871)

```
TPointerArray = Array[0..MaxPtrs-1] of Pointer
```

Array with maxmimum allowed number of pointers

```
TStreamRec = packed record
  ObjType : Sw_Word;
  VmtLink : pointer;
  Load : Pointer;
  Store : Pointer;
  Next : PStreamRec;
end
```

`TSreamRec` is used by the Objects unit streaming mechanism: when an object is registered, a `TStreamRec` record is added to a list of records. This list is used when objects need to be streamed from/streamed to a stream. It contains all the information needed to stream the object.

```
TStrIndex = Array[0..9999] of TStrIndexRec
```

Pointer array type used in a TStringList (922)

```
TStrIndexRec = packed record
  Key : Sw_Word;
  Count : Word;
  Offset : Word;
end
```

Record type used in a TStringList (922) to store the strings

```
TWordArray = Array[0..MaxWords-1] of Word
```

Array with maxmimum allowed number of words.

```
WordRec = packed record
  Hi : Byte;
  Lo : Byte;
end
```

Record describing a Word (in bytes)

21.2.3 Variables

```
invalidhandle : THandle
```

Value for invalid handle. Initial value for file stream handles or when the stream is closed.

21.3 Procedures and functions

21.3.1 Abstract

Synopsis: Abstract error handler.

Declaration: `procedure Abstract`

Visibility: default

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.

21.3.2 CallPointerConstructor

Synopsis: Call a constructor with a pointer argument.

Declaration:
```
function CallPointerConstructor(Ctor: pointer;Obj: pointer;VMT: pointer;
                                Param1: pointer) : pointer
```

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. `Param1` is passed to the constructor. The return value is a pointer to the instance.

Note that this can only be used on constructors that require a pointer as the sole argument. It can also be used to call a constructor with a single argument by reference.

Errors: If the constructor expects other arguments than a pointer, the stack may be corrupted.

See also: CallVoidConstructor (862), CallPointerMethod (861), CallVoidLocal (862), CallPointerLocal (861), CallVoidMethodLocal (863), CallPointerMethodLocal (861)

21.3.3 CallPointerLocal

Synopsis: Call a local nested function with a pointer argument

Declaration:
```
function CallPointerLocal(Func: pointer;Frame: Pointer;Param1: pointer)
                                  : pointer
```

Visibility: default

Description: `CallPointerLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: CallPointerMethod (861), CallVoidMethod (862), CallVoidLocal (862), CallVoidMethodLocal (863), CallPointerMethodLocal (861), CallVoidConstructor (862), CallPointerConstructor (860)

21.3.4 CallPointerMethod

Synopsis: Call a method with a single pointer argument

Declaration:
```
function CallPointerMethod(Method: pointer;Obj: pointer;Param1: pointer)
                                  : pointer
```

Visibility: default

Description: `CallPointerMethod` calls the method with address `Method` for instance `Obj`. It passes `Param1` to the method as the single argument. It returns a pointer to the instance.

Errors: If the method expects other parameters than a single pointer, the stack may become corrupted.

See also: CallVoidMethod (862), CallVoidLocal (862), CallPointerLocal (861), CallVoidMethodLocal (863), CallPointerMethodLocal (861), CallVoidConstructor (862), CallPointerConstructor (860)

21.3.5 CallPointerMethodLocal

Synopsis: Call a local procedure of a method with a pointer argument

Declaration:
```
function CallPointerMethodLocal(Func: pointer;Frame: Pointer;
                                  Obj: pointer;Param1: pointer) : pointer
```

Visibility: default

Description: `CallPointerMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: CallPointerMethod (861), CallVoidMethod (862), CallPointerLocal (861), CallVoidLocal (862), CallVoidMethodLocal (863), CallVoidConstructor (862), CallPointerConstructor (860)

21.3.6 CallVoidConstructor

Synopsis: Call a constructor with no arguments

Declaration:
```
function CallVoidConstructor(Ctor: pointer;Obj: pointer;VMT: pointer)
                                    : pointer
```

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. The return value is a pointer to the instance.

Note that this can only be used on constructors that require no arguments.

Errors: If the constructor expects arguments, the stack may be corrupted.

See also: CallPointerConstructor (860), CallPointerMethod (861), CallVoidLocal (862), CallPointerLocal (861), CallVoidMethodLocal (863), CallPointerMethodLocal (861)

21.3.7 CallVoidLocal

Synopsis: Call a local nested procedure.

Declaration:
```
function CallVoidLocal(Func: pointer;Frame: Pointer) : pointer
```

Visibility: default

Description: `CallVoidLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: CallPointerMethod (861), CallVoidMethod (862), CallPointerLocal (861), CallVoidMethodLocal (863), CallPointerMethodLocal (861), CallVoidConstructor (862), CallPointerConstructor (860)

21.3.8 CallVoidMethod

Synopsis: Call an object method

Declaration:
```
function CallVoidMethod(Method: pointer;Obj: pointer) : pointer
```

Visibility: default

Description: `CallVoidMethod` calls the method with address `Method` for instance `Obj`. It returns a pointer to the instance.

Errors: If the method expects parameters, the stack may become corrupted.

See also: CallPointerMethod (861), CallVoidLocal (862), CallPointerLocal (861), CallVoidMethodLocal (863), CallPointerMethodLocal (861), CallVoidConstructor (862), CallPointerConstructor (860)

21.3.9 CallVoidMethodLocal

Synopsis: Call a local procedure of a method

Declaration:
```
function CallVoidMethodLocal(Func: pointer;Frame: Pointer;Obj: pointer)
                                    : pointer
```

Visibility: default

Description: `CallVoidMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: CallPointerMethod (861), CallVoidMethod (862), CallPointerLocal (861), CallVoidLocal (862), CallPointerMethodLocal (861), CallVoidConstructor (862), CallPointerConstructor (860)

21.3.10 DisposeStr

Synopsis: Dispose of a shortstring which was allocated on the heap.

Declaration:
```
procedure DisposeStr(P: PString)
```

Visibility: default

Description: `DisposeStr` removes a dynamically allocated string from the heap.

For an example, see NewStr (864).

Errors: None.

See also: NewStr (864), SetStr (866)

21.3.11 LongDiv

Synopsis: Overflow safe divide

Declaration:
```
function LongDiv(X: LongInt;Y: Integer) : Integer
```

Visibility: default

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 (863)

21.3.12 LongMul

Synopsis: Overflow safe multiply.

Declaration:
```
function LongMul(X: Integer;Y: Integer) : LongInt
```

Visibility: default

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 (863)

21.3.13 NewStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `function NewStr(const S: String) : PString`

Visibility: default

Description: `NewStr` makes a copy of the string `S` on the heap, and returns a pointer to this copy. If the string is empty then `Nil` is returned.

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 (863), SetStr (866)

Listing: ./objectex/ex40.pp

```
Program ex40;

{ Program to demonstrate the NewStr function }

Uses Objects;

Var S : String;
    P : PString;

begin
  S:='Some really cute string';
  P:=NewStr(S);
  If P^<>S then
    WriteIn ('Oh-oh... Something is wrong !!');
  DisposeStr(P);
end.
```

21.3.14 RegisterObjects

Synopsis: Register standard objects.

Declaration: `procedure RegisterObjects`

Visibility: default

Description: `RegisterObjects` registers the following objects for streaming:

1. `TCollection`, see TCollection (871).
2. `TStringCollection`, see TStringCollection (920).
3. `TStrCollection`, see TStrCollection (910).

Errors: None.

See also: RegisterType (865)

21.3.15 RegisterType

Synopsis: Register new object for streaming.

Declaration: `procedure RegisterType(var S: TStreamRec)`

Visibility: default

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.

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.

The VMT of the object can be retrieved with the following expression:

```
VmtLink: Ofs(TypeOf(MyType)^);
```

Errors: In case of error (if a object with the same `ObjType`) is already registered), run-time error 212 occurs.

Listing: ./objectex/myobject.pp

```
Unit MyObject;


Interface

Uses Objects;

Type
     PMyObject = ^TMyObject;
     TMyObject = Object(TObject)
       Field : Longint;
       Constructor Init;
       Constructor Load (Var Stream : TStream);
       Destructor Done;
       Procedure Store (Var Stream : TStream);
       Function  GetField : Longint;
       Procedure SetField (Value : Longint);
       end;

Implementation

Constructor TMyobject.Init;

begin
  Inherited Init;
  Field:=-1;
end;
```

```
Constructor TMyobject.Load (Var Stream : TStream);

begin
  Stream.Read(Field,Sizeof(Field));
end;

Destructor TMyObject.Done;

begin
end;

Function TMyObject.GetField : Longint;

begin
  GetField:=Field;
end;

Procedure TMyObject.SetField (Value : Longint);

begin
  Field:=Value;
end;

Procedure TMyObject.Store (Var Stream : TStream);

begin
  Stream.Write(Field,SizeOf(Field));
end;

Const MyObjectRec : TStreamRec = (
        Objtype : 666;
        vmtlink : Ofs(TypeOf(TMyObject)^);
        Load : @TMyObject.Load;
        Store : @TMyObject.Store;
        );

begin
  RegisterObjects;
  RegisterType (MyObjectRec);
end.
```

21.3.16 SetStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `procedure SetStr(var p: PString;const s: String)`

Visibility: default

Description: `SetStr` makes a copy of the string `S` on the heap and returns the pointer to this copy in `P`. If `P` pointed to another string (i.e. was not `Nil`, the memory is released first. Contrary to NewStr (864), if the string is empty then a pointer to an empty string is returned.

The allocated memory is not based on the declared size of the string passed to `NewStr`, but is based on the actual length of the string.

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

See also: DisposeStr (863), NewStr (864)

21.4 TBufStream

21.4.1 Description

`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.

21.4.2 Method overview

Page	Property	Description
868	Close	Flush data and Close the file.
868	Done	Close the file and cleans up the instance.
868	Flush	FLush data from buffer, and write it to stream.
867	Init	Initialize an instance of `TBufStream` and open the file.
870	Open	Open the file if it is closed.
870	Read	Read data from the file to a buffer in memory.
869	Seek	Set current position in file.
869	Truncate	Flush buffer, and truncate the file at current position.
870	Write	Write data to the file from a buffer in memory.

21.4.3 TBufStream.Init

Synopsis: Initialize an instance of `TBufStream` and open the file.

Declaration:
```
constructor Init(FileName: FNameStr;Mode: Word;Size: Word)
```

Visibility: default

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.

For an example see TBufStream.Flush (868).

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

See also: TDosStream.Init (886), TBufStream.Done (868)

21.4.4 TBufStream.Done

Synopsis: Close the file and cleans up the instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` flushes and closes the file if it was open and cleans up the instance of `TBufStream`.

For an example see TBufStream.Flush (868).

Errors: None.

See also: TDosStream.Done (886), TBufStream.Init (867), TBufStream.Close (868)

21.4.5 TBufStream.Close

Synopsis: Flush data and Close the file.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` flushes and closes the file if it was open, and sets `Handle` to -1. Contrary to Done (868) it does not clean up the instance of `TBufStream`

For an example see TBufStream.Flush (868).

Errors: None.

See also: TStream.Close (916), TBufStream.Init (867), TBufStream.Done (868)

21.4.6 TBufStream.Flush

Synopsis: FLush data from buffer, and write it to stream.

Declaration: `procedure Flush; Virtual`

Visibility: default

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 (870) for more info on the errors.

See also: TStream.Close (916), TBufStream.Init (867), TBufStream.Done (868)

Listing: ./objectex/ex15.pp

```
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.
```

21.4.7 TBufStream.Truncate

Synopsis: Flush buffer, and truncate the file at current position.

Declaration: `procedure Truncate;  Virtual`

Visibility: default

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.

For an example, see TDosStream.Truncate (887).

Errors: Errors can be those of Flush (868) or TDosStream.Truncate (887).

See also: TStream.Truncate (917), TDosStream.Truncate (887), TStream.GetSize (914)

21.4.8 TBufStream.Seek

Synopsis: Set current position in file.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

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.

For an example, see TStream.Seek (918);

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 (918), TStream.GetPos (914)

21.4.9 TBufStream.Open

Synopsis: Open the file if it is closed.

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

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 (868) call.

For an example, see TDosStream.Open (889).

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 (916), TBufStream.Close (868)

21.4.10 TBufStream.Read

Synopsis: Read data from the file to a buffer in memory.

Declaration: `procedure Read(var Buf;Count: LongInt);  Virtual`

Visibility: default

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.

For an example, see TStream.Read (919).

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 (919), TBufStream.Write (870)

21.4.11 TBufStream.Write

Synopsis: Write data to the file from a buffer in memory.

Declaration: `procedure Write(var Buf;Count: LongInt);  Virtual`

Visibility: default

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.

For an example, see TStream.Read (919).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: TStream.Write (919), TBufStream.Read (870)

21.5 TCollection

21.5.1 Description

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.

21.5.2 Method overview

Page	Property	Description
873	At	Return the item at a certain index.
881	AtDelete	Delete item at certain position.
880	AtFree	Free an item at the indicates position, calling it's destructor.
884	AtInsert	Insert an element at a certain position in the collection.
884	AtPut	Set collection item, overwriting an existing value.
880	Delete	Delete an item from the collection, but does not destroy it.
878	DeleteAll	Delete all elements from the collection. Objects are not destroyed.
872	Done	Clean up collection, release all memory.
883	Error	Set error code.
875	FirstThat	Return first item which matches a test.
882	ForEach	Execute procedure for each item in the list.
879	Free	Free item from collection, calling it's destructor.
877	FreeAll	Release all objects from the collection.
881	FreeItem	Destroy a non-nil item.
874	GetItem	Read one item off the stream.
873	IndexOf	Find the position of a certain item.
871	Init	Instantiate a new collection.
879	Insert	Insert a new item in the collection at the end.
875	LastThat	Return last item which matches a test.
872	Load	Initialize a new collection and load collection from a stream.
876	Pack	Remove all `>Nil` pointers from the collection.
885	PutItem	Put one item on the stream
883	SetLimit	Set maximum number of elements in the collection.
885	Store	Write collection to a stream.

21.5.3 TCollection.Init

Synopsis: Instantiate a new collection.

Declaration: `constructor Init(ALimit: Sw_Integer;ADelta: Sw_Integer)`

Visibility: default

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.

For an example, see TCollection.ForEach (882).

Errors: None.

See also: TCollection.Load (872), TCollection.Done (872)

21.5.4 TCollection.Load

Synopsis: Initialize a new collection and load collection from a stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

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 (874).

See also: TCollection.Init (871), TCollection.GetItem (874), TCollection.Done (872)

Listing: ./objectex/ex22.pp

```
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;
  WriteIn ('Inserted ',C^.Count,' objects');
  S:=New(PMemorySTream,Init(1000,10));
  C^.Store(S^);
  C^.FreeAll;
  Dispose(C,Done);
  S^.Seek(0);
  C^.Load(S^);
  WriteIn ('Read ',C^.Count,' objects from stream.');
  Dispose(S,Done);
  Dispose(C,Done);
end.
```

21.5.5 TCollection.Done

Synopsis: Clean up collection, release all memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` frees all objects in the collection, and then releases all memory occupied by the instance.

For an example, see TCollection.ForEach (882).

Errors: None.

See also: TCollection.Init (871), TCollection.FreeAll (877)

21.5.6 TCollection.At

Synopsis: Return the item at a certain index.

Declaration: `function At(Index: Sw_Integer) : Pointer`

Visibility: default

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, seepl{Error}{TCollection.Error} is called with `coIndexError` and `Index` as arguments, resulting in a run-time error.

See also: TCollection.Insert (879)

Listing: ./objectex/ex23.pp

```
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);
    WriteIn ('Object ',i,' has field : ',M^.GetField);
    end;
  C^.FreeAll;
  Dispose(C,Done);
end.
```

21.5.7 TCollection.IndexOf

Synopsis: Find the position of a certain item.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. If `Item` isn't present in the collection, -1 is returned.

Errors: If the item is not present, -1 is returned.

See also: TCollection.At (873), TCollection.GetItem (874), TCollection.Insert (879)

Listing: ./objectex/ex24.pp

```
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.
```

21.5.8 TCollection.GetItem

Synopsis: Read one item off the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

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 (912).

See also: TStream.Get (912), TCollection.Store (885)

21.5.9 TCollection.LastThat

Synopsis: Return last item which matches a test.

Declaration: `function LastThat(Test: Pointer) : Pointer`

Visibility: default

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: TCollection.FirstThat (875)

Listing: ./objectex/ex25.pp

```
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.
```

21.5.10 TCollection.FirstThat

Synopsis: Return first item which matches a test.

Declaration: `function FirstThat(Test: Pointer) : Pointer`

Visibility: default

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: TCollection.LastThat (875)

Listing: ./objectex/ex26.pp

```
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.
```

21.5.11 TCollection.Pack

Synopsis: Remove all `>Nil` pointers from the collection.

Declaration: `procedure Pack`

Visibility: default

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: TCollection.SetLimit (883)

Listing: ./objectex/ex26.pp

```
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.
```

21.5.12 TCollection.FreeAll

Synopsis: Release all objects from the collection.

Declaration: `procedure FreeAll`

Visibility: default

Description: `FreeAll` calls the destructor of each object in the collection. It doesn't release any memory occupied by the collection itself, but it does set `Count` to zero.

Errors:

See also: TCollection.DeleteAll (878), TCollection.FreeItem (881)

Listing: ./objectex/ex28.pp

```
Program ex28;

{ Program to demonstrate the TCollection.FreeAll method }

Uses Objects,MyObject; { For TMyObject definition and registration }
```

```
Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  C^.FreeAll;
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

21.5.13 TCollection.DeleteAll

Synopsis: Delete all elements from the collection. Objects are not destroyed.

Declaration: `procedure DeleteAll`

Visibility: default

Description: `DeleteAll` deletes all elements from the collection. It just sets the `Count` variable to zero. Contrary to FreeAll (877), `DeletAll` doesn't call the destructor of the objects.

Errors: None.

See also: TCollection.FreeAll (877), TCollection.Delete (880)

Listing: ./objectex/ex29.pp

```
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 : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
```

```
    end;
  WriteLn ('Added 100 Items.');
  C^.DeleteAll;
  WriteLn ('Deleted all objects.');
  Dispose(C,Done);
end.
```

21.5.14 TCollection.Free

Synopsis: Free item from collection, calling it's destructor.

Declaration: `procedure Free(Item: Pointer)`

Visibility: default

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: TCollection.FreeItem (881)

Listing: ./objectex/ex30.pp

```
Program ex30;

{ Program to demonstrate the TCollection.Free method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  WriteLn ('Added 100 Items.');
  With C^ do
    While Count>0 do Free(At(Count-1));
  WriteLn ('Freed all objects.');
  Dispose(C,Done);
end.
```

21.5.15 TCollection.Insert

Synopsis: Insert a new item in the collection at the end.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

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: TCollection.AtInsert (884), TCollection.AtPut (884)

21.5.16 TCollection.Delete

Synopsis: Delete an item from the collection, but does not destroy it.

Declaration: `procedure Delete(Item: Pointer)`

Visibility: default

Description: `Delete` deletes `Item` from the collection. It doesn't call the item's destructor, though. For this the Free (879) call is provided.

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

See also: TCollection.AtDelete (881), TCollection.Free (879)

Listing: ./objectex/ex31.pp

```
Program ex31;

{ Program to demonstrate the TCollection.Delete method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do Delete(At(Count-1));
  Writeln ('Freed all objects');
  Dispose(C,Done);
end.
```

21.5.17 TCollection.AtFree

Synopsis: Free an item at the indicates position, calling it's destructor.

Declaration: `procedure AtFree(Index: Sw_Integer)`

Visibility: default

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 (883) is called with `CoIndexError`.

See also: TCollection.Free (879), TCollection.AtDelete (881)

Listing: ./objectex/ex32.pp

```
Program ex32;

{ Program to demonstrate the TCollection.AtFree method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items');
  With C^ do
    While Count>0 do AtFree(Count-1);
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

21.5.18 TCollection.FreeItem

Synopsis: Destroy a non-nil item.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `FreeItem` calls the destructor of `Item` if it is not nil.

Remark: This function is used internally by the TCollection object, and should not be called directly.

Errors: None.

See also: TCollection.Free (879), TCollection.AtFree (880)

21.5.19 TCollection.AtDelete

Synopsis: Delete item at certain position.

Declaration: `procedure AtDelete(Index: Sw_Integer)`

Visibility: default

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 (883) is called with `CoIndexError`.

See also: TCollection.Delete (880)

Listing: ./objectex/ex33.pp

```
Program ex33;

{ Program to demonstrate the TCollection.AtDelete method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do AtDelete(Count-1);
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

21.5.20 TCollection.ForEach

Synopsis: Execute procedure for each item in the list.

Declaration: `procedure ForEach(Action: Pointer)`

Visibility: default

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: TCollection.FirstThat (875), TCollection.LastThat (875)

Listing: ./objectex/ex21.pp

```
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
  WriteIn ('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;
  WriteIn ('Inserted ',C^.Count,' objects');
  C^.ForEach(@PrintField);
  C^.FreeAll;
  Dispose(C,Done);
end.
```

21.5.21 TCollection.SetLimit

Synopsis: Set maximum number of elements in the collection.

Declaration: `procedure SetLimit(ALimit: Sw_Integer);  Virtual`

Visibility: default

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`

For an example, see Pack (876).

Errors: None.

See also: TCollection.Init (871)

21.5.22 TCollection.Error

Synopsis: Set error code.

Declaration: `procedure Error(Code: Integer;Info: Integer);  Virtual`

Visibility: default

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 (860)

21.5.23 TCollection.AtPut

Synopsis: Set collection item, overwriting an existing value.

Declaration: `procedure AtPut(Index: Sw_Integer;Item: Pointer)`

Visibility: default

Description: `AtPut` sets the element at position `Index` in the collection to `Item`. Any previous value is overwritten.

For an example, see Pack (876).

Errors: If `Index` isn't valid then Error (883) is called with `CoIndexError`.

21.5.24 TCollection.AtInsert

Synopsis: Insert an element at a certain position in the collection.

Declaration: `procedure AtInsert(Index: Sw_Integer;Item: Pointer)`

Visibility: default

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

See also: TCollection.Insert (879)

Listing: ./objectex/ex34.pp

```
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.
```

21.5.25 TCollection.Store

Synopsis: Write collection to a stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

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 (872).

For an example, see TCollection.Load (872).

Errors: Errors returned are those by TStream.Put (917).

See also: TCollection.Load (872), TCollection.PutItem (885)

21.5.26 TCollection.PutItem

Synopsis: Put one item on the stream

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

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 (917).

See also: TCollection.Store (885), TCollection.GetItem (874)

21.6 TDosStream

21.6.1 Description

`TDosStream` is a stream that stores it's contents in a file. it overrides a couple of methods of TStream (912) for this.

In addition to the fields inherited from `TStream` (see TStream (912)), 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 TBufStream (867).

21.6.2 Method overview

Page	Property	Description
887	Close	Close the file.
886	Done	Closes the file and cleans up the instance.
886	Init	Instantiate a new instance of TDosStream.
889	Open	Open the file stream
889	Read	Read data from the stream to a buffer.
888	Seek	Set file position.
887	Truncate	Truncate the file on the current position.
890	Write	Write data from a buffer to the stream.

21.6.3 TDosStream.Init

Synopsis: Instantiate a new instance of TDosStream.

Declaration: `constructor Init(FileName: FNameStr;Mode: Word)`

Visibility: default

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.

For an example, see TDosStream.Truncate (887).

Errors: On error, Status (**??**) is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: TDosStream.Done (886)

21.6.4 TDosStream.Done

Synopsis: Closes the file and cleans up the instance.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` closes the file if it was open and cleans up the instance of `TDosStream`.

for an example, see e.g. TDosStream.Truncate (887).

Errors: None.

See also: TDosStream.Init (886), TDosStream.Close (887)

21.6.5 TDosStream.Close

Synopsis: Close the file.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` closes the file if it was open, and sets `Handle` to -1. Contrary to Done (886) it does not clean up the instance of `TDosStream`

For an example, see TDosStream.Open (889).

Errors: None.

See also: TStream.Close (916), TDosStream.Init (886), TDosStream.Done (886)

21.6.6 TDosStream.Truncate

Synopsis: Truncate the file on the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

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 (917), TStream.GetSize (914)

Listing: ./objectex/ex16.pp

```
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.
```

21.6.7 TDosStream.Seek

Synopsis: Set file position.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

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 (918), TStream.GetPos (914)

Listing: ./objectex/ex17.pp

```
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.
```

21.6.8 TDosStream.Open

Synopsis: Open the file stream

Declaration: `procedure Open(OpenMode: Word); Virtual`

Visibility: default

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 (887) 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 (916), TDosStream.Close (887)

Listing: ./objectex/ex14.pp

```
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.
```

21.6.9 TDosStream.Read

Synopsis: Read data from the stream to a buffer.

Declaration: `procedure Read(var Buf;Count: LongInt); Virtual`

Visibility: default

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`.

For an example, see TStream.Read (919).

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 (919), TDosStream.Write (890)

21.6.10 TDosStream.Write

Synopsis: Write data from a buffer to the stream.

Declaration: `procedure Write(var Buf;Count: LongInt);  Virtual`

Visibility: default

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.

For an example, see TStream.Read (919).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: TStream.Write (919), TDosStream.Read (889)

21.7 TMemoryStream

21.7.1 Description

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.

21.7.2 Method overview

Page	Property	Description
891	Done	Clean up memory and destroy the object instance.
890	Init	Initialize memory stream, reserves memory for stream data.
892	Read	Read data from the stream to a location in memory.
891	Truncate	Set the stream size to the current position.
892	Write	Write data to the stream.

21.7.3 TMemoryStream.Init

Synopsis: Initialize memory stream, reserves memory for stream data.

Declaration: `constructor Init(ALimit: LongInt;ABlockSize: Word)`

Visibility: default

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.

For an example, see e.g TStream.CopyFrom (920).

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: TMemoryStream.Done (891)

21.7.4 TMemoryStream.Done

Synopsis: Clean up memory and destroy the object instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` releases the memory blocks used by the stream, and then cleans up the memory used by the stream object itself.

For an example, see e.g TStream.CopyFrom (920).

Errors: None.

See also: TMemoryStream.Init (890)

21.7.5 TMemoryStream.Truncate

Synopsis: Set the stream size to the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

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 (917)

Listing: ./objectex/ex20.pp

```
Program ex20;

{ Program to demonstrate the TMemoryStream.Truncate method }

Uses Objects;

Var L : String;
    P : PString;
    S : PMemoryStream;
    I : Longint;

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PMemoryStream,Init(1000,100));
  Writeln ('Writing 100 times "',L,'" to stream.');
  For I:=1 to 100 do
    S^.WriteStr(@L);
  Writeln ('Finished.');
  S^.Seek(100);
  S^.Truncate;
  Writeln ('Truncated at byte 100.');
  Dispose (S,Done);
  Writeln ('Finished.');
end.
```

21.7.6 TMemoryStream.Read

Synopsis: Read data from the stream to a location in memory.

Declaration: `procedure Read(var Buf;Count: LongInt); Virtual`

Visibility: default

Description: `Read` reads `Count` bytes from the stream to `Buf`. It updates the position of the stream.

For an example, see TStream.Read (919).

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 (919), TMemoryStream.Write (892)

21.7.7 TMemoryStream.Write

Synopsis: Write data to the stream.

Declaration: `procedure Write(var Buf;Count: LongInt); Virtual`

Visibility: default

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 (890)) as needed.

For an example, see TStream.Read (919).

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

See also: TStream.Write (919), TMemoryStream.Read (892)

21.8 TObject

21.8.1 Description

This type serves as the basic object for all other objects in the Objects unit.

21.8.2 Method overview

Page	Property	Description
894	Done	Destroy an object.
893	Free	Destroy an object and release all memory.
892	Init	Construct (initialize) a new object
893	Is_Object	Check whether a pointer points to an object.

21.8.3 TObject.Init

Synopsis: Construct (initialize) a new object

Declaration: `constructor Init`

Visibility: default

Description: Instantiates a new object of type `TObject`. It fills the instance up with Zero bytes.

For an example, see Free (893)

Errors: None.

See also: TObject.Free (893), TObject.Done (894)

21.8.4 TObject.Free

Synopsis: Destroy an object and release all memory.

Declaration: `procedure Free`

Visibility: default

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: TObject.Init (892), TObject.Done (894)

Listing: ./objectex/ex7.pp

```
program ex7;

{ Program to demonstrate the TObject.Free call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O:=New(PObject,Init);
  // Free memory of object.
  O^.free;
end.
```

21.8.5 TObject.Is_Object

Synopsis: Check whether a pointer points to an object.

Declaration: `function Is_Object(P: Pointer) : Boolean`

Visibility: default

Description: `Is_Object` returns `True` if the pointer `P` points to an instance of a `TObject` descendent, it returns `false` otherwise.

21.8.6 TObject.Done

Synopsis: Destroy an object.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done`, the destructor of `TObject` does nothing. It is mainly intended to be used in the TObject.Free (893) method.

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

Errors: None.

See also: TObject.Free (893), TObject.Init (892)

Listing: ./objectex/ex8.pp

```
program ex8;

{ Program to demonstrate the TObject.Done call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O:=New(PObject,Init);
  O^.Done;
end.
```

21.9 TPoint

21.9.1 Description

Record describing a point in a 2 dimensional plane.

21.10 TRect

21.10.1 Description

Describes a rectangular region in a plane.

21.10.2 Method overview

Page	Property	Description
899	Assign	Set rectangle corners.
896	Contains	Determine if a point is inside the rectangle
896	Copy	Copy cornerpoints from another rectangle.
895	Empty	Is the surface of the rectangle zero
896	Equals	Do the corners of the rectangles match
899	Grow	Expand rectangle with certain size.
897	Intersect	Reduce rectangle to intersection with another rectangle
898	Move	Move rectangle along a vector.
897	Union	Enlarges rectangle to encompas another rectangle.

21.10.3 TRect.Empty

Synopsis: Is the surface of the rectangle zero

Declaration: `function Empty : Boolean`

Visibility: default

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 (896), TRect.Contains (896)

Listing: ./objectex/ex1.pp

```
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.
```

21.10.4 TRect.Equals

Synopsis: Do the corners of the rectangles match

Declaration: `function Equals(R: TRect) : Boolean`

Visibility: default

Description: `Equals` returns `True` if the rectangle has the same corner points `A,B` as the rectangle R, and `False` otherwise.

For an example, see TRect.Empty (895)

Errors: None.

See also: TRect.Empty (895), TRect.Contains (896)

21.10.5 TRect.Contains

Synopsis: Determine if a point is inside the rectangle

Declaration: `function Contains(P: TPoint) : Boolean`

Visibility: default

Description: `Contains` returns `True` if the point `P` is contained in the rectangle (including borders), `False` otherwise.

Errors: None.

See also: TRect.Intersect (897), TRect.Equals (896)

21.10.6 TRect.Copy

Synopsis: Copy cornerpoints from another rectangle.

Declaration: `procedure Copy(R: TRect)`

Visibility: default

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: TRect.Assign (899)

Listing: ./objectex/ex2.pp

```
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
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
end.
```

21.10.7 TRect.Union

Synopsis: Enlarges rectangle to encompas another rectangle.

Declaration: `procedure Union(R: TRect)`

Visibility: default

Description: `Union` enlarges the current rectangle so that it becomes the union of the current rectangle with the rectangle `R`.

Errors: None.

See also: TRect.Intersect (897)

Listing: ./objectex/ex3.pp

```
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
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
end.
```

21.10.8 TRect.Intersect

Synopsis: Reduce rectangle to intersection with another rectangle

Declaration: `procedure Intersect(R: TRect)`

Visibility: default

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: TRect.Union (897)

Listing: ./objectex/ex4.pp

```
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
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
  BRect.Assign(25,25,30,30);
  Arect.Intersect(BRect);
  If ARect.Empty Then
    WriteIn ('ARect is empty');
end.
```

21.10.9 TRect.Move

Synopsis: Move rectangle along a vector.

Declaration: `procedure Move(ADX: Sw_Integer;ADY: Sw_Integer)`

Visibility: default

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: TRect.Grow (899)

Listing: ./objectex/ex5.pp

```
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
    WriteIn ('ARect equals BRect')
  Else
    WriteIn ('ARect does not equal BRect !');
end.
```

21.10.10 TRect.Grow

Synopsis: Expand rectangle with certain size.

Declaration: `procedure Grow(ADX: Sw_Integer;ADY: Sw_Integer)`

Visibility: default

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: TRect.Move (898)

Listing: ./objectex/ex6.pp

```
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
    WriteIn ('ARect equals BRect')
  Else
    WriteIn ('ARect does not equal BRect !');
end.
```

21.10.11 TRect.Assign

Synopsis: Set rectangle corners.

Declaration: `procedure Assign(XA: Sw_Integer;YA: Sw_Integer;XB: Sw_Integer; YB: Sw_Integer)`

Visibility: default

Description: `Assign` sets the corner points of the rectangle to `(XA,YA)` and `(Xb,Yb)`.

For an example, see TRect.Copy (896).

Errors: None.

See also: TRect.Copy (896)

21.11 TResourceCollection

21.11.1 Description

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: Remark that the `TResourceCollection` manages the names of the resources and their assiciated positions and sizes, it doesn't manage the resources themselves.

21.11.2 Method overview

Page	Property	Description
901	FreeItem	Release memory occupied by item.
901	GetItem	Read an item from the stream.
900	KeyOf	Return the key of an item in the collection.
901	PutItem	Write an item to the stream.

21.11.3 TResourceCollection.KeyOf

Synopsis: Return the key of an item in the collection.

Declaration: `function KeyOf(Item: Pointer) : Pointer;  Virtual`

Visibility: default

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 (921)

21.11.4 TResourceCollection.GetItem

Synopsis: Read an item from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

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 (872) method.

Errors: Errors returned are those by TStream.Read (919)

See also: TCollection.Load (872), TStream.Read (919)

21.11.5 TResourceCollection.FreeItem

Synopsis: Release memory occupied by item.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

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 (881)

21.11.6 TResourceCollection.PutItem

Synopsis: Write an item to the stream.

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

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 (885) method.

Errors: Errors returned are those by TStream.Write (919).

See also: TCollection.Store (885)

21.12 TResourceFile

21.12.1 Description

TResourceFile (901) represents the resources in a binary file image.

21.12.2 Method overview

Page	Property	Description
902	Count	Number of resources in the file
904	Delete	Delete a resource from the file
902	Done	Destroy the instance and remove it from memory.
903	Flush	Writes the resources to the stream.
903	Get	Return a resource by key name.
902	Init	Instantiate a new instance.
903	KeyAt	Return the key of the item at a certain position.
904	Put	Set a resource by key name.
903	SwitchTo	Write resources to a new stream.

21.12.3 TResourceFile.Init

Synopsis: Instantiate a new instance.

Declaration: `constructor Init(AStream: PStream)`

Visibility: default

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: TResourceFile.Done (902)

21.12.4 TResourceFile.Done

Synopsis: Destroy the instance and remove it from memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

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: TResourceFile.Init (902)

21.12.5 TResourceFile.Count

Synopsis: Number of resources in the file

Declaration: `function Count : Sw_Integer`

Visibility: default

Description: `Count` returns the number of resources. If no resources were read, zero is returned.

Errors: None.

See also: TResourceFile.Init (902)

21.12.6 TResourceFile.KeyAt

Synopsis: Return the key of the item at a certain position.

Declaration: `function KeyAt(I: Sw_Integer) : String`

Visibility: default

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: TResourceFile.Get (903)

21.12.7 TResourceFile.Get

Synopsis: Return a resource by key name.

Declaration: `function Get(Key: String) : PObject`

Visibility: default

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`

21.12.8 TResourceFile.SwitchTo

Synopsis: Write resources to a new stream.

Declaration: `function SwitchTo(AStream: PStream;Pack: Boolean) : PStream`

Visibility: default

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 (919) and TStream.Write (919).

See also: TResourceFile.Flush (903)

21.12.9 TResourceFile.Flush

Synopsis: Writes the resources to the stream.

Declaration: `procedure Flush`

Visibility: default

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 (918) and TStream.Write (919).

See also: TResourceFile.SwitchTo (903)

21.12.10 TResourceFile.Delete

Synopsis: Delete a resource from the file

Declaration: `procedure Delete(Key: String)`

Visibility: default

Description: `Delete` deletes the resource identified by `Key` from the collection. It sets the `Modified` flag to true.

Errors: None.

See also: TResourceFile.Flush (903)

21.12.11 TResourceFile.Put

Synopsis: Set a resource by key name.

Declaration: `procedure Put(Item: PObject;Key: String)`

Visibility: default

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 (917) and `TStream.Seek`

See also: TResourceFile.Get (903)

21.13 TSortedCollection

21.13.1 Description

`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 (906) method.

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

The Compare (906) 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.

The example below defines a descendent of `TSortedCollection` which is used in the examples.

21.13.2 Method overview

Page	Property	Description
906	Compare	Compare two items in the collection.
906	IndexOf	Return index of an item in the collection.
905	Init	Instantiates a new instance of a `TSortedCollection`
908	Insert	Insert new item in collection.
905	KeyOf	Return the key of an item
905	Load	Instantiates a new instance of a `TSortedCollection` and loads it from stream.
907	Search	Search for item with given key.
909	Store	Write the collection to the stream.

21.13.3 TSortedCollection.Init

Synopsis: Instantiates a new instance of a `TSortedCollection`

Declaration: `constructor Init(ALimit: Sw_Integer;ADelta: Sw_Integer)`

Visibility: default

Description: `Init` calls the inherited constuctor (see TCollection.Init (871)) 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: TSortedCollection.Load (905), TCollection.Done (872)

21.13.4 TSortedCollection.Load

Synopsis: Instantiates a new instance of a `TSortedCollection` and loads it from stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: `Load` calls the inherited constuctor (see TCollection.Load (872)) 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.

For an example, see TCollection.Load (872).

Errors: None.

See also: TSortedCollection.Init (905), TCollection.Done (872)

21.13.5 TSortedCollection.KeyOf

Synopsis: Return the key of an item

Declaration: `function KeyOf(Item: Pointer) : Pointer;  Virtual`

Visibility: default

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: TSortedCollection.IndexOf (906), TSortedCollection.Compare (906)

21.13.6 TSortedCollection.IndexOf

Synopsis: Return index of an item in the collection.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer;  Virtual`

Visibility: default

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.

For an example, see TCollection.IndexOf (873)

Errors: None.

See also: TSortedCollection.Search (907), TSortedCollection.Compare (906)

21.13.7 TSortedCollection.Compare

Synopsis: Compare two items in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

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 (907) method and in the Insert (908) 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: TSortedCollection.IndexOf (906), TSortedCollection.Search (907)

Listing: ./objectex/mysortc.pp

```
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.
```

21.13.8 TSortedCollection.Search

Synopsis: Search for item with given key.

Declaration: `function Search(Key: Pointer;var Index: Sw_Integer) : Boolean;  Virtual`

Visibility: default

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 (906) function to implement this search.

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

Errors: None.

See also: TCollection.IndexOf (873)

Listing: ./objectex/ex36.pp

```
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
  WriteIn ('Field : ',P^.GetField);
end;


begin
  Randomize;
  C:=New(PMySortedCollection,Init(120,10));
  C^.Duplicates:=True;
  WriteIn ('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
        WriteIn ('No such value found')
      else
        begin
        Write ('Value ',PMyObject(C^.At(I))^.GetField);
        WriteIn (' present at position ',I);
        end;
      end;
  Until I=-1;
  Dispose(M,Done);
  Dispose(C,Done);
end.
```

21.13.9 TSortedCollection.Insert

Synopsis: Insert new item in collection.

Declaration: `procedure Insert(Item: Pointer);  Virtual`

Visibility: default

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 (884), 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: TCollection.AtInsert (884)

Listing: ./objectex/ex35.pp

```
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.
```

21.13.10 TSortedCollection.Store

Synopsis: Write the collection to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection to the stream `S`. It does this by calling the inherited TCollection.Store (885), and then writing the `Duplicates` flag to the stream.

After a `Store`, the collection can be loaded from the stream with the constructor Load (905)

For an example, see TCollection.Load (872).

Errors: Errors can be those of TStream.Put (917).

See also: TSortedCollection.Load (905)

21.14 TStrCollection

21.14.1 Description

The `TStrCollection` object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the Compare (906) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

21.14.2 Method overview

Page	Property	Description
910	Compare	Compare two strings in the collection.
911	FreeItem	Free null-terminated string from the collection.
911	GetItem	Read a null-terminated string from the stream.
911	PutItem	Write a null-terminated string to the stream.

21.14.3 TStrCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

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 (906)

Listing: ./objectex/ex38.pp

```
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
        WriteIn ('Duplicate string found at position ',I);
  Dispose(C,Done);
end.
```

21.14.4 TStrCollection.GetItem

Synopsis: Read a null-terminated string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

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 (913).

See also: TStrCollection.PutItem (911)

21.14.5 TStrCollection.FreeItem

Synopsis: Free null-terminated string from the collection.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `TStrCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: TCollection.FreeItem (881)

21.14.6 TStrCollection.PutItem

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

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 (918).

See also: TStrCollection.GetItem (911)

21.15 TStream

21.15.1 Description

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 descendent types.

Programs should not instantiate objects of type TStream directly, but instead instantiate a descendant type, such as `TDosStream`, `TMemoryStream`.

21.15.2 Method overview

Page	Property	Description
916	Close	Close the stream
920	CopyFrom	Copy data from another stream.
918	Error	Set stream status
917	Flush	Flush the stream data from the buffer, if any.
912	Get	Read an object definition from the stream.
914	GetPos	Return current position in the stream
914	GetSize	Return the size of the stream.
912	Init	Constructor for TStream instance
916	Open	Open the stream
917	Put	Write an object to the stream.
919	Read	Read data from stream to buffer.
915	ReadStr	Read a shortstring from the stream.
916	Reset	Reset the stream
918	Seek	Set stream position.
913	StrRead	Read a null-terminated string from the stream.
918	StrWrite	Write a null-terminated string to the stream.
917	Truncate	Truncate the stream size on current position.
919	Write	Write a number of bytes to the stream.
918	WriteStr	Write a pascal string to the stream.

21.15.3 TStream.Init

Synopsis: Constructor for TStream instance

Declaration: `constructor Init`

Visibility: default

Description: `Init` initializes a TStream instance. Descendent streams should always call the inherited `Init`.

21.15.4 TStream.Get

Synopsis: Read an object definition from the stream.

Declaration: `function Get : PObject`

Visibility: default

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: TStream.Put (917)

Listing: ./objectex/ex9.pp

```
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.
```

21.15.5 TStream.StrRead

Synopsis: Read a null-terminated string from the stream.

Declaration: `function StrRead : PChar`

Visibility: default

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: TStream.StrWrite (918), TStream.ReadStr (915)

Listing: ./objectex/ex10.pp

```
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.
```

21.15.6 TStream.GetPos

Synopsis: Return current position in the stream

Declaration: `function GetPos : LongInt;  Virtual`

Visibility: default

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: TStream.Seek (918), TStream.GetSize (914)

Listing: ./objectex/ex11.pp

```
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.
```

21.15.7 TStream.GetSize

Synopsis: Return the size of the stream.

Declaration: `function GetSize : LongInt;  Virtual`

Visibility: default

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: TStream.Seek (918), TStream.GetPos (914)

Listing: ./objectex/ex12.pp

```
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.
```

21.15.8 TStream.ReadStr

Synopsis: Read a shortstring from the stream.

Declaration: `function ReadStr : PString`

Visibility: default

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: TStream.StrRead (913)

Listing: ./objectex/ex13.pp

```
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.
```

21.15.9 TStream.Open

Synopsis: Open the stream

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

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.

For an example, see TDosStream.Open (889).

Errors: None.

See also: TStream.Close (916), TStream.Reset (916)

21.15.10 TStream.Close

Synopsis: Close the stream

Declaration: `procedure Close;  Virtual`

Visibility: default

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.

for an example, see TDosStream.Open (889).

Errors: None.

See also: TStream.Open (916), TStream.Reset (916)

21.15.11 TStream.Reset

Synopsis: Reset the stream

Declaration: `procedure Reset`

Visibility: default

Description: `Reset` sets the stream's status to `0`, as well as the ErrorInfo

Errors: None.

See also: TStream.Open (916), TStream.Close (916)

21.15.12 TStream.Flush

Synopsis: Flush the stream data from the buffer, if any.

Declaration: `procedure Flush; Virtual`

Visibility: default

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.

for an example, see TBufStream.Flush (868).

Errors: None.

See also: TStream.Truncate (917)

21.15.13 TStream.Truncate

Synopsis: Truncate the stream size on current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

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.

For an example, see TDosStream.Truncate (887).

Errors: None.

See also: TStream.Seek (918)

21.15.14 TStream.Put

Synopsis: Write an object to the stream.

Declaration: `procedure Put(P: PObject)`

Visibility: default

Description: `Put` writes the object pointed to by `P.P` should be non-nil. The object type must have been registered with RegisterType (865).

After the object has been written, it can be read again with Get (912).

For an example, see TStream.Get (912);

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: TStream.Get (912)

21.15.15 TStream.StrWrite

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure StrWrite(P: PChar)`

Visibility: default

Description: `StrWrite` writes the null-terminated string `P` to the stream. `P` can only be 65355 bytes long.

For an example, see TStream.StrRead (913).

Errors: None.

See also: TStream.WriteStr (918), TStream.StrRead (913), TStream.ReadStr (915)

21.15.16 TStream.WriteStr

Synopsis: Write a pascal string to the stream.

Declaration: `procedure WriteStr(P: PString)`

Visibility: default

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

For an example, see TStream.ReadStr (915).

Errors: None.

See also: TStream.StrWrite (918), TStream.StrRead (913), TStream.ReadStr (915)

21.15.17 TStream.Seek

Synopsis: Set stream position.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

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)

For an example, see TDosStream.Seek (888).

Errors: If `Pos` is larger than the stream size, `Status` is set to `StSeekError`.

See also: TStream.GetPos (914), TStream.GetSize (914)

21.15.18 TStream.Error

Synopsis: Set stream status

Declaration: `procedure Error(Code: Integer;Info: Integer);  Virtual`

Visibility: default

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.

21.15.19 TStream.Read

Synopsis: Read data from stream to buffer.

Declaration: `procedure Read(var Buf;Count: LongInt); Virtual`

Visibility: default

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: TStream.Write (919), TStream.ReadStr (915), TStream.StrRead (913)

Listing: ./objectex/ex18.pp

```
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
      WriteIn ('Buffer differs at position ',I);
  Dispose(S,Done);
end.
```

21.15.20 TStream.Write

Synopsis: Write a number of bytes to the stream.

Declaration: `procedure Write(var Buf;Count: LongInt); Virtual`

Visibility: default

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`.

For an example, see TStream.Read (919).

Errors: No checking is done to see if `Buf` actually contains `Count` bytes.

See also: TStream.Read (919), TStream.WriteStr (918), TStream.StrWrite (918)

21.15.21 TStream.CopyFrom

Synopsis: Copy data from another stream.

Declaration: `procedure CopyFrom(var S: TStream;Count: LongInt)`

Visibility: default

Description: `CopyFrom` reads Count bytes from stream `S` and stores them in the current stream. It uses the Read (919) method to read the data, and the Write (919) method to write in the current stream.

Errors: None.

See also: TStream.Read (919), TStream.Write (919)

Listing: ./objectex/ex19.pp

```
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.
```

21.16 TStringCollection

21.16.1 Description

The `TStringCollection` object manages a sorted collection of pascal strings. To this end, it overrides the Compare (906) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

21.16.2 Method overview

Page	Property	Description
921	Compare	Compare two strings in the collection.
922	FreeItem	Dispose a string in the collection from memory.
921	GetItem	Get string from the stream.
922	PutItem	Write a string to the stream.

21.16.3 TStringCollection.GetItem

Synopsis: Get string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

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 (915).

See also: TStringCollection.PutItem (922)

21.16.4 TStringCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `TStringCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns the following results:

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

0 if the two strings are equal.

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

Errors: None.

See also: TSortedCollection.Compare (906)

Listing: ./objectex/ex37.pp

```
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.
```

21.16.5 TStringCollection.FreeItem

Synopsis: Dispose a string in the collection from memory.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `TStringCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: TCollection.FreeItem (881)

21.16.6 TStringCollection.PutItem

Synopsis: Write a string to the stream.

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

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 (918).

See also: TStringCollection.GetItem (921)

21.17 TStringList

21.17.1 Description

A `TStringList` object can be used to read a collection of strings stored in a stream. If you register this object with the RegisterType (865) function, you cannot register the `TStrListMaker` object.

21.17.2 Method overview

Page	Property	Description
923	Done	Clean up the instance
923	Get	Return a string by key name
923	Load	Load stringlist from stream.

21.17.3 TStringList.Load

Synopsis: Load stringlist from stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: The `Load` constructor reads the `TStringList` object from the stream `S`. It also reads the descriptions of the strings from the stream. The string descriptions are stored as an array of `TstrIndexrec` records, where each record describes a string on the stream. These records are kept in memory.

Errors: If an error occurs, a stream error is triggered.

See also: TStringList.Done (923)

21.17.4 TStringList.Done

Synopsis: Clean up the instance

Declaration: `destructor Done; Virtual`

Visibility: default

Description: The `Done` destructor frees the memory occupied by the string descriptions, and destroys the object.

Errors: None.

See also: TStringList.Load (923), TObject.Done (894)

21.17.5 TStringList.Get

Synopsis: Return a string by key name

Declaration: `function Get(Key: Sw_Word) : String`

Visibility: default

Description: `Get` reads the string with key `Key` from the list of strings on the stream, and returns this string. If there is no string with such a key, an empty string is returned.

Errors: If no string with key `Key` is found, an empty string is returned. A stream error may result if the stream doesn't contain the needed strings.

See also: TStrListMaker.Put (924)

21.18 TStrListMaker

21.18.1 Description

The `TStrListMaker` object can be used to generate a stream with strings, which can be read with the `TStringList` object. If you register this object with the RegisterType (865) function, you cannot register the `TStringList` object.

21.18.2 Method overview

Page	Property	Description
924	Done	Clean up the instance and free all related memory.
924	Init	Instantiate a new instance of `TStrListMaker`
924	Put	Add a new string to the list with associated key.
925	Store	Write the strings to the stream.

21.18.3 TStrListMaker.Init

Synopsis: Instantiate a new instance of `TStrListMaker`

Declaration: `constructor Init(AStrSize: Sw_Word;AIndexSize: Sw_Word)`

Visibility: default

Description: The `Init` constructor creates a new instance of the `TstrListMaker` object. It allocates `AStrSize` bytes on the heap to hold all the strings you wish to store. It also allocates enough room for `AIndexSize` key description entries (of the type `TStrIndexrec`).

`AStrSize` must be large enough to contain all the strings you wish to store. If not enough memory is allocated, other memory will be overwritten. The same is true for `AIndexSize` : maximally `AIndexSize` strings can be written to the stream.

Errors: None.

See also: TObject.Init (892), TStrListMaker.Done (924)

21.18.4 TStrListMaker.Done

Synopsis: Clean up the instance and free all related memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: The `Done` destructor de-allocates the memory for the index description records and the string data, and then destroys the object.

Errors: None.

See also: TObject.Done (894), TStrListMaker.Init (924)

21.18.5 TStrListMaker.Put

Synopsis: Add a new string to the list with associated key.

Declaration: `procedure Put(Key: Sw_Word;S: String)`

Visibility: default

Description: `Put` adds they string `S` with key `Key` to the collection of strings. This action doesn't write the string to a stream. To write the strings to the stream, see the Store (925) method.

Errors: None.

See also: TStrListMaker.Store (925)

21.18.6 TStrListMaker.Store

Synopsis: Write the strings to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection of strings to the stream `S`. The collection can then be read with the `TStringList` object.

Errors: A stream error may occur when writing the strings to the stream.

See also: TStringList.Load (923), TStrListMaker.Put (924)

21.19 TUnSortedStrCollection

21.19.1 Description

The `TUnSortedStrCollection` object manages an unsorted list of strings. To this end, it overrides the TStringCollection.Insert (920) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the Search (907) and IndexOf (873) methods will not work on an unsorted string collection.

21.19.2 Method overview

Page	Property	Description
925	Insert	Insert a new string in the collection.

21.19.3 TUnSortedStrCollection.Insert

Synopsis: Insert a new string in the collection.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

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: None.

See also: TCollection.Insert (879)

Listing: ./objectex/ex39.pp

```
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;
    C^.Insert(NewStr(S));
    end;
  For I:=0 to 99 do
    Writeln (I:2,': ',PString(C^.At(i))^ );
  Dispose(C,Done);
end.
```

Chapter 22

Reference for unit 'objpas'

22.1 Overview

The objpas unit is meant for compatibility with Object Pascal as implemented by Delphi. The unit is loaded automatically by the Free Pascal compiler whenever the `Delphi` or `objfpc` more is entered, either through the command line switches `-Sd` or `-Sh` or with the `{$MODE DELPHI}` or `{$MODE OBJFPC}` directives.

It redefines some basic pascal types, introduces some functions for compatibility with Delphi's system unit, and introduces some methods for the management of the resource string tables.

22.2 Constants, types and variables

22.2.1 Constants

```
MaxInt = MaxLongint
```

Maximum value for Integer (927) type.

22.2.2 Types

```
Integer = LongInt
```

In `OBJPAS` mode and in `DELPHI` mode, an `Integer` has a size of 32 bit. In TP or regular FPC mode, an integer is 16 bit.

```
IntegerArray = Array[0..$efffffff] of Integer
```

Generic array of integer (927)

```
PInteger = ^Integer
```

Pointer to Integer (927) type.

```
PIntegerArray = ^IntegerArray
```

Pointer to TIntegerArray (928) type.

```
PointerArray = Array[0..512*1024*1024-2] of Pointer
```

Generic Array of pointers.

```
PPointerArray = ^PointerArray
```

Pointer to PointerArray (928)

```
PResStringRec = ^AnsiString
```

Pointer to ansistring (Delphi compatibility).

```
PString = PAnsiString
```

Pointer to ansistring type.

```
TBoundArray = Array of Integer
```

Array of integer, used in interfaces.

```
TIntegerArray = IntegerArray
```

Alias for IntegerArray (927)

```
TPointerArray = PointerArray
```

Alias for PointerArray (928)

```
TResourceIterator = function(Name: AnsiString;Value: AnsiString;
                             Hash: LongInt;arg: pointer) : AnsiString
```

The resource string tables can be managed with a callback function which the user must provide: `TResourceIterator`.

```
TResStringRec = AnsiString
```

Ansistring record in resource table (Delphi compatibility).

22.2.3 Variables

```
ExceptionClass : TClass
```

`ExceptionClass` is the base class for all exceptions. Normally, this is Exception (1496), definied in the Sysutils (1350) unit.

```
ExceptObjProc : Pointer
```

`ExceptObjProc` is unused in Free Pascal, and is provided for Delphi compatibility only.

22.3 Procedures and functions

22.3.1 AssignFile

Synopsis: Assign text or untyped file

Declaration:
```
procedure AssignFile(var f: File;const Name: String)
procedure AssignFile(var f: File;p: pchar)
procedure AssignFile(var f: File;c: Char)
procedure AssignFile(var t: Text;const s: String)
procedure AssignFile(var t: Text;p: pchar)
procedure AssignFile(var t: Text;c: Char)
procedure AssignFile(var f: TypedFile;const Name: String)
procedure AssignFile(var f: TypedFile;p: pchar)
procedure AssignFile(var f: TypedFile;c: Char)
```

Visibility: default

Description: `AssignFile` is completely equivalent to the system unit's Assign (1210) function: It assigns `Name` to a function of any type (`FileType` can be `Text` or a typed or untyped `File` variable). `Name` can be a string, a single character or a `PChar`.

It is most likely introduced to avoid confusion between the regular Assign (1210) function and the `Assign` method of `TPersistent` in the Delphi VCL.

Errors: None.

See also: CloseFile (929), #rtl.system.Assign (1210), #rtl.system.Reset (1302), #rtl.system.Rewrite (1303), #rtl.system.Append (1208)

Listing: ./refex/ex88.pp

```
Program Example88;

{ Program to demonstrate the AssignFile and CloseFile functions. }

{$MODE Delphi}

Var F : text;

begin
  AssignFile(F,'textfile.tmp');
  Rewrite(F);
  Writeln (F,'This is a silly example of AssignFile and CloseFile.');
  CloseFile(F);
end.
```

22.3.2 CloseFile

Synopsis: Close text or untyped file

Declaration:
```
procedure CloseFile(var f: File)
procedure CloseFile(var t: Text)
```

Visibility: default

Description: `CloseFile` flushes and closes a file `F` of any file type. `F` can be `Text` or a typed or untyped `File` variable. After a call to `CloseFile`, any attempt to write to the file `F` will result in an error.

It is most likely introduced to avoid confusion between the regular Close (1217) function and the `Close` method of `TForm` in the Delphi VCL.

for an example, see AssignFile (929).

Errors: None.

See also: #rtl.system.Close (1217), AssignFile (929), #rtl.system.Reset (1302), #rtl.system.Rewrite (1303), #rtl.system.Append (1208)

22.3.3 FinalizeResourceTables

Synopsis: Finalize resource string tables

Declaration: `procedure FinalizeResourceTables`

Visibility: default

Description: `FinalizeResourceTables` should be called by any routine that sets the resource string tables to translated versions of the strings. Typically, it should be called in the `Finalization` code of the unit that does the initialization of the tables. This is needed to correctly deallocate the strings from memory

22.3.4 GetResourceStringCurrentValue

Synopsis: Return current value of resourcestring

Declaration:
```
function GetResourceStringCurrentValue(TableIndex: LongInt;
                                       StringIndex: LongInt) : AnsiString
```

Visibility: default

Description: `GetResourceStringCurrentValue` returns the current value of the resourcestring in table `TableIndex` with index `StringIndex`.

The current value depends on the system of internationalization that was used, and which language is selected when the program is executed.

Errors: If either `TableIndex` or `StringIndex` are out of range, then a empty string is returned.

See also: SetResourceStrings (935), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934), ResourceStringCount (934)

Listing: ./refex/ex90.pp

```
Program Example90;

{ Program to demonstrate the GetResourceStringCurrentValue function. }
{$Mode Delphi}

ResourceString

  First  = 'First string';
  Second = 'Second String';

Var I,J : Longint;
```

```
begin
  { Print current values of all resourcestrings }
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      WriteIn (I,',',J,' : ',GetResourceStringCurrentValue(I,J));
end.
```

22.3.5 GetResourceStringDefaultValue

Synopsis: Return default (original) value of resourcestring

Declaration: `function GetResourceStringDefaultValue(TableIndex: LongInt; StringIndex: LongInt) : AnsiString`

Visibility: default

Description: `GetResourceStringDefaultValue` returns the default value of the resourcestring in table `TableIndex` with index `StringIndex`.

The default value is the value of the string that appears in the source code of the programmer, and is compiled into the program.

Errors: If either `TableIndex` or `StringIndex` are out of range, then a empty string is returned.

See also: SetResourceStrings (935), GetResourceStringCurrentValue (930), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934), ResourceStringCount (934)

Listing: ./refex/ex91.pp

```
Program Example91;

{ Program to demonstrate the GetResourceStringDefaultValue function. }
{$Mode Delphi}

ResourceString

  First  = 'First string';
  Second = 'Second String';

Var I,J : Longint;

begin
  { Print default values of all resourcestrings }
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      WriteIn (I,',',J,' : ',GetResourceStringDefaultValue(I,J));
end.
```

22.3.6 GetResourceStringHash

Synopsis: Return hash value of resource string

Declaration: `function GetResourceStringHash(TableIndex: LongInt;StringIndex: LongInt) : LongInt`

Visibility: default

Description: `GetResourceStringHash` returns the hash value associated with the resource string in table `TableIndex`, with index `StringIndex`.

The hash value is calculated from the default value of the resource string in a manner that gives the same result as the GNU gettext mechanism. It is stored in the resourcestring tables, so retrieval is faster than actually calculating the hash for each string.

For an example, see Hash (933).

Errors: If either `TableIndex` or `StringIndex` is zero, 0 is returned.

See also: Hash (933), SetResourceStrings (935), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934), ResourceStringCount (934)

22.3.7 GetResourceStringName

Synopsis: Return name of resource string.

Declaration:
```
function GetResourceStringName(TableIndex: LongInt;StringIndex: LongInt)
                                        : Ansistring
```

Visibility: default

Description: `GetResourceStringName` returns the name of the resourcestring in table `TableIndex` with index `StringIndex`. The name of the string is always the unit name in which the string was declared, followed by a period and the name of the constant, all in lowercase.

If a unit MyUnit declares a resourcestring `MyTitle` then the name returned will be `myunit.mytitle`. A resourcestring in the program file will have the name of the program prepended.

The name returned by this function is also the name that is stored in the resourcestring file generated by the compiler.

Strictly speaking, this information isn't necessary for the functioning of the program, it is provided only as a means to easier translation of strings.

Errors: If either `TableIndex` or `StringIndex` is zero, an empty string is returned.

See also: SetResourceStrings (935), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934), ResourceStringCount (934)

Listing: ./refex/ex92.pp

```
Program Example92;

{ Program to demonstrate the GetResourceStringName function. }
{$Mode Delphi}

ResourceString

  First  = 'First string';
  Second = 'Second String';

Var I,J : Longint;

begin
  { Print names of all resourcestrings }
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      Writeln (I,',',J,' : ',GetResourceStringName(I,J));
end.
```

22.3.8 Hash

Synopsis: Create GNU Gettext hash value for a string

Declaration: `function Hash(S: AnsiString) : LongWord`

Visibility: default

Description: `Hash` calculates the hash value of the string `S` in a manner that is compatible with the GNU gettext hash value for the string. It is the same value that is stored in the Resource string tables, and which can be retrieved with the GetResourceStringHash (931) function call.

Errors: None. In case the calculated hash value should be 0, the returned result will be -1.

See also: GetResourceStringHash (931)

Listing: ./refex/ex93.pp

```
Program Example93;

{ Program to demonstrate the Hash function. }
{$Mode Delphi}

ResourceString

  First  = 'First string';
  Second = 'Second String';

Var I,J : Longint;

begin
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      If Hash(GetResourceStringDefaultValue(I,J))
        <>GetResourceStringHash(I,J) then
        WriteLn ('Hash mismatch at ',I,',',J)
      else
        WriteLn ('Hash (',I,',',J,') matches.');
end.
```

22.3.9 LoadResString

Synopsis: Load resource string

Declaration: `function LoadResString(p: PResStringRec) : AnsiString`

Visibility: default

22.3.10 ParamStr

Synopsis: Return command-line parameter

Declaration: `function ParamStr(Param: Integer) : Ansistring`

Visibility: default

Description: `ParamStr` returns the `Param`-th command-line parameter as an AnsiString. The system unit Paramstr (933) function limits the result to 255 characters, and is overridden with this function.

The zeroeth command-line parameter contains the path of the executable. On some operating systems (BSD) it may be simply the command as typed on the command-line, because the OS does not offer a method to retrieve the full binary name.

For an example, see #rtl.system.Paramstr (1294).

Errors: In case `Param` is an invalid value, an empty string is returned.

See also: Paramstr (933)

22.3.11 ResetResourceTables

Synopsis: Restore all resource strings to their declared values

Declaration: `procedure ResetResourceTables`

Visibility: default

Description: `ResetResourceTables` resets all resource strings to their default (i.e. as in the source code) values.

Normally, this should never be called from a user's program. It is called in the initialization code of the objpas unit. However, if the resourcetables get messed up for some reason, this procedure will fix them again.

Errors: None.

See also: SetResourceStrings (935), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934), ResourceStringCount (934)

22.3.12 ResourceStringCount

Synopsis: Return number of resource strins in table

Declaration: `function ResourceStringCount(TableIndex: LongInt) : LongInt`

Visibility: default

Description: `ResourceStringCount` returns the number of resourcestrings in the table with index `TableIndex`. The strings in a particular table are numbered from `0` to `ResourceStringCount-1`, i.e. they're zero based.

For an example, see GetResourceStringDefaultValue (931)

Errors: If an invalid `TableIndex` is given, `-1` is returned.

See also: SetResourceStrings (935), GetResourceStringCurrentValue (930), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934)

22.3.13 ResourceStringTableCount

Synopsis: Return number of resource string tables

Declaration: `function ResourceStringTableCount : LongInt`

Visibility: default

Description: `ResourceStringTableCount` returns the number of resource string tables; this may be zero if no resource strings are used in a program.

The tables are numbered from 0 to `ResourceStringTableCount-1`, i.e. they're zero based.

For an example, see GetResourceStringDefaultValue (931)

Errors:

See also: SetResourceStrings (935), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringCount (934)

22.3.14 SetResourceStrings

Synopsis: Set values of all resource strings.

Declaration:
```
procedure SetResourceStrings(SetFunction: TResourceIterator;
                             arg: pointer)
```

Visibility: default

Description: `SetResourceStrings` calls `SetFunction` for all resourcestrings in the resourcestring tables and sets the resourcestring's current value to the value returned by `SetFunction`.

The `Name`,`Value` and `Hash` parameters passed to the iterator function are the values stored in the tables.

Errors: None.

See also: GetResourceStringCurrentValue (930), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934), ResourceStringCount (934)

Listing: ./refex/ex95.pp

```
Program Example95;

{ Program to demonstrate the SetResourceStrings function. }
{$Mode objfpc}

ResourceString

  First  = 'First string';
  Second = 'Second String';

Var I,J : Longint;
    S : AnsiString;

Function Translate (Name,Value : AnsiString; Hash : longint): AnsiString;

begin
  WriteLn ('Translate (',Name,') => ',Value);
  Write   ('->');
  ReadLn  (Result);
end;

begin
  SetResourceStrings(@Translate);
  WriteLn ('Translated strings : ');
  For I:=0 to ResourceStringTableCount-1 do
```

```
    For J:=0 to ResourceStringCount(i)-1 do
      begin
      WriteIn (GetResourceStringDefaultValue(I,J));
      WriteIn ('Translates to : ');
      WriteIn (GetResourceStringCurrentValue(I,J));
      end;
end.
```

22.3.15 SetResourceStringValue

Synopsis: Set value of a resource string

Declaration:
```
function SetResourceStringValue(TableIndex: LongInt;
                                StringIndex: LongInt;Value: Ansistring)
                                 : Boolean
```

Visibility: default

Description: `SetResourceStringValue` assigns `Value` to the resource string in table `TableIndex` with index `StringIndex`.

Errors:

See also: SetResourceStrings (935), GetResourceStringCurrentValue (930), GetResourceStringDefaultValue (931), GetResourceStringHash (931), GetResourceStringName (932), ResourceStringTableCount (934), ResourceStringCount (934)

Listing: ./refex/ex94.pp

```
Program Example94;

{ Program to demonstrate the SetResourceStringValue function. }
{$Mode Delphi}

ResourceString

  First  = 'First string';
  Second = 'Second String';

Var I,J : Longint;
    S : AnsiString;

begin
  { Print current values of all resourcestrings }
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      begin
      WriteIn ('Translate => ',GetResourceStringDefaultValue(I,J));
      Write   ('->');
      ReadIn(S);
      SetResourceStringValue(I,J,S);
      end;
  WriteIn ('Translated strings : ');
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      begin
      WriteIn (GetResourceStringDefaultValue(I,J));
```

```
      WriteLn ('Translates to : ');
      WriteLn (GetResourceStringCurrentValue(I,J));
      end;
end.
```

22.3.16 SetUnitResourceStrings

Synopsis: Set unit resource strings for a given unit

Declaration:
```
procedure SetUnitResourceStrings(const UnitName: String;
                                 SetFunction: TResourceIterator;
                                 arg: pointer)
```

Visibility: default

Description: `SetUnitResourceStrings` will call `SetFunction` for all resource strings of the unit indicated by `UnitName`. The additional `Arg` pointer will be passed to the `SetFunction` iterator.

Errors: None.

See also: SetResourceStrings (935), ResetResourceTables (934), FinalizeResourceTables (930)

Chapter 23

Reference for unit 'oldlinux'

23.1 Utility routines

Auxiliary functions that are useful in connection with the other functions.

Table 23.1:

Name	Description
CreateShellArgV (999)	Create an array of pchars from string
EpochToLocal (1002)	Convert epoch time to local time
FD_Clr (1013)	Clear item of select filedescriptors
FD_IsSet (1013)	Check item of select filedescriptors
FD_Set (1013)	Set item of select filedescriptors
FD_ZERO (1014)	Clear all items in select filedecriptors
LocalToEpoch (1032)	Convert local time to epoch time
MMap (1035)	Map a file into memory
MUnMap (1036)	Unmap previously mapped memory file
Octal (1038)	Convert octal to digital
S_ISBLK (1054)	Check file mode for block device
S_ISCHR (1055)	Check file mode for character device
S_ISDIR (1055)	Check file mode for directory
S_ISFIFO (1055)	Check file mode for FIFO
S_ISLNK (1055)	Check file mode for symboloc link
S_ISREG (1056)	Check file mode for regular file
S_ISSOCK (1056)	Check file mode for socket
StringToPPchar (1051)	Create an array of pchars from string

23.2 Terminal functions

Functions for controlling the terminal to which the process is connected.

23.3 System information

Functions for retrieving system information such as date and time.

Table 23.2:

Name	Description
CFMakeRaw (994)	Set terminal to raw mode
CFSetISpeed (994)	Set terminal reading speed
CFSetOSpeed (994)	Set terminal writing speed
IOCtl (1029)	General IO control call
IsATTY (1031)	See if filedescriptor is a terminal
TCDrain (1057)	Wait till all output was written
TCFlow (1057)	Suspend transmission or receipt of data
TCFlush (1057)	Discard data written to terminal
TCGetAttr (1058)	Get terminal attributes
TCGetPGrp (1058)	Return PID of foreground process
TCSendBreak (1059)	Send data for specific time
TCSetAttr (1059)	Set terminal attributes
TCSetPGrp (1060)	Set foreground process
TTYName (1060)	Name of tty file

Table 23.3:

Name	Description
GetDate (1020)	Return system date
GetDateTime (1020)	Return system date and time
GetDomainName (1021)	Return system domain name
GetEpochTime (1022)	Return epoch time
GetHostName (1024)	Return system host name
GetLocalTimezone (1025)	Return system timezone
GetTime (1027)	Return system time
GetTimeOfDay (1027)	Return system time
GetTimezoneFile (1028)	Return name of timezone file
ReadTimezoneFile (1043)	Read timezone file contents
SysInfo (1053)	Return general system information
Uname (1061)	Return system information

23.4 Signals

Functions for managing and responding to signals.

23.5 Process handling

Functions for managing processes and programs.

23.6 Directory handling routines

Functions for reading and searching directories.

Table 23.4:

Name	Description
Alarm (990)	Send alarm signal to self
Kill (1031)	Send arbitrary signal to process
pause (1040)	Wait for signal to arrive
SigAction (1047)	Set signal action
Signal (1048)	Set signal action
SigPending (1049)	See if signals are waiting
SigProcMask (1049)	Set signal processing mask
SigRaise (1050)	Send signal to self
SigSuspend (1051)	Sets signal mask and waits for signal
NanoSleep (1037)	Waits for a specific amount of time

23.7 Pipes, FIFOs and streams

Functions for creating and managing pipes.

23.8 General File handling routines

Functions for handling files on disk.

23.9 File Input/Output routines

Functions for handling file input/output.

23.10 Overview

This document describes the LINUX unit for Free Pascal. The unit was written by Michael van Canneyt. It works only on the Linux/X86 operating system.

23.11 Constants, types and variables

23.11.1 Constants

```
B0 = $0000000
```

```
B110 = $0000003
```

```
B115200 = $0001002
```

```
B1200 = $0000009
```

Table 23.5:

Name	Description
Clone (997)	Create a thread
Execl (1003)	Execute process with command-line list
Execle (1004)	Execute process with command-line list and environment
Execlp (1004)	Search in path and execute process with command list
Execv (1005)	Execute process
Execve (1006)	Execute process with environment
Execvp (1007)	Search in path and execute process
Fork (1016)	Spawn child process
GetEGid (1021)	Get effective group id
GetEnv (1022)	Get environment variable
GetEUid (1023)	Get effective user id
GetGid (1024)	Get group id
GetPid (1025)	Get process id
GetPPid (1026)	Get parent process id
GetPriority (1026)	Get process priority
GetUid (1028)	Get user id
Nice (1038)	Change priority of process
SetPriority (1046)	Change priority of process
Shell (1046)	Execute shell command
WaitPid (1063)	Wait for child process to terminate

Table 23.6:

Name	Description
CloseDir (999)	Close directory handle
Glob (1028)	Return files matching a search expression
GlobFree (1029)	Free result of Glob
OpenDir (1039)	Open directory for reading
ReadDir (1041)	Read directory entry
SeekDir (1043)	Seek directory
TellDir (1060)	Seek directory

```
B134 = $0000004

B150 = $0000005

B1800 = $000000A

B19200 = $000000E

B200 = $0000006

B230400 = $0001003
```

Table 23.7:

Name	Description
AssignPipe (991)	Create a pipe
AssignStream (992)	Create pipes to program's input and output
MkFifo (1035)	Make a fifo
PClose (1040)	Close a pipe
POpen (1040)	Open a pipe for to program's input or output

Table 23.8:

Name	Description
Access (989)	Check access rights on file
BaseName (993)	Return name part of file
Chown (996)	Change owner of file
Chmod (995)	Change access rights on file
DirName (1000)	Return directory part of file
FSplit (1017)	Split filename in parts
FExpand (1014)	Return full-grown filename
FLock (1014)	Set lock on a file
FNMatch (1015)	Match filename to searchpattern
FSearch (1017)	Search for a file in a path
FSStat (1018)	Return filesystem information
FStat (1019)	Return file information
FRename (1016)	Rename file
LStat (1033)	Return information on a link
Link (1031)	Create a link
ReadLink (1042)	Read contents of a symbolic link
SymLink (1052)	Create a symbolic link
Umask (1061)	Set the file creation mask
UnLink (1061)	Remove a file
Utime (1062)	Change file timestamps

```
B2400 = $000000B
```

```
B300 = $0000007
```

```
B38400 = $000000F
```

```
B460800 = $0001004
```

```
B4800 = $000000C
```

```
B50 = $0000001
```

```
B57600 = $0001001
```

Table 23.9:

Name	Description
Dup (1001)	Duplicate a file handle
Dup2 (1001)	Copy one file handle to another
Fcntl (1008)	General file control
fdClose (1009)	Close file descriptor
fdFlush (1009)	Flush file descriptor
fdOpen (1010)	Open new file descriptor
fdRead (1011)	Read from file descriptor
fdSeek (1012)	Position in file
fdTruncate (1012)	Truncate file
fdWrite (1013)	Write to file descriptor
GetFS (1023)	Get file descriptor of pascal file
Select (1043)	Wait for input from file descriptor
SelectText (1045)	Wait for input from pascal file

```
B600 = $0000008
```

```
B75 = $0000002
```

```
B9600 = $000000D
```

```
BRKINT = $0000002
```

```
BS0 = $0000000
```

```
BS1 = $0002000
```

```
BSDLY = $0002000
```

```
CBAUD = $000100F
```

```
CBAUDEX = $0001000
```

```
CIBAUD = $100F0000
```

```
CLOCAL = $0000800
```

```
CLONE_FILES = $00000400
```

Clone (997) option: open files shared between processes

```
CLONE_FS = $00000200
```

Clone (997) option: fs info shared between processes

```
CLONE_PID = $00001000
```

Clone (997) option: PID shared between processes

```
CLONE_SIGHAND = $00000800
```

Clone (997) option: signal handlers shared between processes

```
CLONE_VM = $00000100
```

Clone (997) option: VM shared between processes

```
CMSPAR = $40000000

CR0 = $0000000

CR1 = $0000200

CR2 = $0000400

CR3 = $0000600

CRDLY = $0000600

CREAD = $0000080

CRTSCTS = $80000000

CS5 = $0000000

CS6 = $0000010

CS7 = $0000020

CS8 = $0000030

CSIGNAL = $000000ff
```

Clone (997) option: Signal mask to be sent at exit

```
CSIZE = $0000030
```

```
CSTOPB = $0000040
```

```
ECHO = $0000008
```

```
ECHOCTL = $0000200
```

```
ECHOE = $0000010
```

```
ECHOK = $0000020
```

```
ECHOKE = $0000800
```

```
ECHONL = $0000040
```

```
ECHOPRT = $0000400
```

```
EXTA = B19200
```

```
EXTB = B38400
```

```
FF0 = $0000000
```

```
FF1 = $0008000
```

```
FFDLY = $0008000
```

```
FIOASYNC = $5452
```

```
FIOCLEX = $5451
```

```
FIONBIO = $5421
```

```
FIONCLEX = $5450
```

```
FIONREAD = $541B
```

```
FLUSHO = $0001000
```

```
fs_ext = $137d
```

File system type (FSStat (1018)): (ext) Extended

```
fs_ext2 = $ef53
```

File system type (FSStat (1018)): (ext2) Second extended

```
fs_iso = $9660
```

File system type (FSStat (1018)): ISO 9660

```
fs_minix = $137f
```

File system type (FSStat (1018)): Minix

```
fs_minix_30 = $138f
```

File system type (FSStat (1018)): Minix 3.0

```
fs_minux_V2 = $2468
```

File system type (FSStat (1018)): Minix V2

```
fs_msdos = $4d44
```

File system type (FSStat (1018)): MSDOS (FAT)

```
fs_nfs = $6969
```

File system type (FSStat (1018)): NFS

```
fs_old_ext2 = $ef51
```

File system type (FSStat (1018)): (ext2) Old second extended

```
fs_proc = $9fa0
```

File system type (FSStat (1018)): PROC fs

```
fs_xia = $012FD16D
```

File system type (FSStat (1018)): XIA

```
F_GetFd = 1
```

FCntl (1008) command: Get close-on-exec flag

```
F_GetFl = 3
```

FCntl (1008) command: Get filedescriptor flags

```
F_GetLk = 5
```

FCntl (1008) command: Get lock

```
F_GetOwn = 9
```

FCntl (1008) command: get owner of filedescriptor events

```
F_OK = 0
```

Access (989) call test: file exists.

```
F_SetFd = 2
```

FCntl (1008) command: Set close-on-exec flag

```
F_SetFl = 4
```

FCntl (1008) command: Set filedescriptor flags

```
F_SetLk = 6
```

FCntl (1008) command: Set lock

```
F_SetLkW = 7
```

FCntl (1008) command: Test lock

```
F_SetOwn = 8
```

FCntl (1008) command: Set owner of filedescriptor events

```
HUPCL = $0000400
```

```
ICANON = $0000002
```

```
ICRNL = $0000100
```

```
IEXTEN = $0008000
```

```
IGNBRK = $0000001
```

```
IGNCR = $0000080
```

```
IGNPAR = $0000004
```

```
IMAXBEL = $0002000
```

```
INLCR = $0000040
```

```
INPCK = $0000010
```

```
IOCtl_TCGETS = $5401
```

IOCTL call number: get Terminal Control settings

```
ISIG = $0000001
```

```
ISTRIP = $0000020
```

```
IUCLC = $0000200
```

```
IXANY = $0000800
```

```
IXOFF = $0001000
```

```
IXON = $0000400
```

```
LOCK_EX = 2
```

FLock (1014) Exclusive lock

```
LOCK_NB = 4
```

FLock (1014) Non-blocking operation

```
LOCK_SH = 1
```

FLock (1014) Shared lock

```
LOCK_UN = 8
```

FLock (1014) unlock

```
MAP_ANONYMOUS = $20
```

MMap (1035) map type: Don't use a file

```
MAP_DENYWRITE = $800
```

MMap (1035) option: Ignored.

```
MAP_EXECUTABLE = $1000
```

MMap (1035) option: Ignored.

```
MAP_FIXED = $10
```

MMap (1035) map type: Interpret addr exactly

```
MAP_GROWSDOWN = $100
```

MMap (1035) option: Memory grows downward (like a stack)

```
MAP_LOCKED = $2000
```

MMap (1035) option: lock the pages in memory.

```
MAP_NORESERVE = $4000
```

MMap (1035) option: Do not reserve swap pages for this memory.

```
MAP_PRIVATE = 2
```

MMap (1035) map type: Changes are private

```
MAP_SHARED = $1
```

MMap (1035) map type: Share changes

```
MAP_TYPE = $f
```

MMap (1035) map type: Bitmask for type of mapping

```
MINSIGSTKSZ = 2048
```

```
NCC = 8
```

Number of control characters in termio (985) record.

```
NCCS = 32
```

Number of control characters in termios (986) record.

```
NL0 = $0000000
```

```
NL1 = $0000100
```

```
NLDLY = $0000100
```

```
NOFLSH = $0000080
```

```
OCRNL = $0000008
```

```
OFDEL = $0000080
```

```
OFILL = $0000040
```

```
OLCUC = $0000002
```

```
ONLCR = $0000004
```

```
ONLRET = $0000020
```

```
ONOCR = $0000010
```

```
Open_Accmode = 3
```

Bitmask to determine access mode in open flags.

```
Open_Append = 2 shl 9
```

File open mode: Append to file

```
Open_Creat = 1 shl 6
```

File open mode: Create if file does not yet exist.

```
Open_Direct = 4 shl 12
```

File open mode: Minimize caching effects

```
Open_Directory = 2 shl 15
```

File open mode: File must be directory.

```
Open_Excl = 2 shl 6
```

File open mode: Open exclusively

```
Open_LargeFile = 1 shl 15
```

File open mode: Open for 64-bit I/O

```
Open_NDelay = Open_NonBlock
```

File open mode: Alias for Open_NonBlock (951)

```
Open_NoCtty = 4 shl 6
```

File open mode: No TTY control.

```
Open_NoFollow = 4 shl 15
```

File open mode: Fail if file is symbolic link.

```
Open_NonBlock = 4 shl 9
```

File open mode: Open in non-blocking mode

```
Open_RdOnly = 0
```

File open mode: Read only

```
Open_RdWr = 2
```

File open mode: Read/Write

```
Open_Sync = 1 shl 12
```

File open mode: Write to disc at once

```
Open_Trunc = 1 shl 9
```

File open mode: Truncate file to length 0

```
Open_WrOnly = 1
```

File open mode: Write only

```
OPOST = $0000001

PARENB = $0000100

PARMRK = $0000008

PARODD = $0000200
```

```
PENDIN = $0004000
```

```
Prio_PGrp = 1
```

Get/set process group priority

```
Prio_Process = 0
```

Get/Set process priority

```
Prio_User = 2
```

Get/set user priority

```
PROT_EXEC = $4
```

MMap (1035) memory access: page can be executed

```
PROT_NONE = $0
```

MMap (1035) memory access: page can not be accessed

```
PROT_READ = $1
```

MMap (1035) memory access: page can be read

```
PROT_WRITE = $2
```

MMap (1035) memory access: page can be written

```
P_IN = 1
```

Input file descriptor of pipe pair.

```
P_OUT = 2
```

Output file descriptor of pipe pair.

```
R_OK = 4
```

Access (989) call test: read allowed

```
SA_INTERRUPT = $20000000
```

Sigaction options: ?

```
SA_NOCLDSTOP = 1
```

Sigaction options: Do not receive notification when child processes stop

```
SA_NOMASK = $40000000
```

Sigaction options: Do not prevent the signal from being received when it is handled.

```
SA_ONESHOT = $80000000
```

Sigaction options: Restore the signal action to the default state.

```
SA_ONSTACK = SA_STACK
```

Socket option

```
SA_RESTART = $10000000
```

Sigaction options: Provide behaviour compatible with BSD signal semantics

```
SA_SHIRQ = $04000000
```

Sigaction options: ?

```
SA_STACK = $08000000
```

Sigaction options: Call the signal handler on an alternate signal stack.

```
Seek_Cur = 1
```

Seek option: Set position relative to current position.

```
Seek_End = 2
```

Seek option: Set position relative to end of file.

```
Seek_set = 0
```

Seek option: Set absolute position.

```
SIGABRT = 6
```

Signal: ABRT (Abort)

```
SIGALRM = 14
```

Signal: ALRM (Alarm clock)

```
SIGBUS = 7
```

Signal: BUS (bus error)

```
SIGCHLD = 17
```

Signal: CHLD (child status changed)

```
SIGCONT = 18
```

Signal: CONT (Continue)

```
SIGFPE = 8
```

Signal: FPE (Floating point error)

```
SIGHUP = 1
```

Signal: HUP (Hangup)

```
SIGILL = 4
```

Signal: ILL (Illegal instruction)

```
SIGINT = 2
```

Signal: INT (Interrupt)

```
SIGIO = 29
```

Signal: IO (I/O operation possible)

```
SIGIOT = 6
```

Signal: IOT (IOT trap)

```
SIGKILL = 9
```

Signal: KILL (unblockable)

```
SIGPIPE = 13
```

Signal: PIPE (Broken pipe

```
SIGPOLL = SIGIO
```

Signal: POLL (Pollable event)

```
SIGPROF = 27
```

Signal: PROF (Profiling alarm)

```
SIGPWR = 30
```

Signal: PWR (power failure restart)

```
SIGQUIT = 3
```

Signal: QUIT

```
SIGSEGV = 11
```

Signal: SEGV (Segmentation violation)

```
SIGSTKFLT = 16
```

Signal: STKFLT (Stack Fault)

```
SIGSTKSZ = 8192
```

Signal Stack size error

```
SIGSTOP = 19
```

Signal: STOP (Stop, unblockable)

```
SIGTerm = 15
```

Signal: TERM (Terminate)

```
SIGTRAP = 5
```

Signal: TRAP (Trace trap)

```
SIGTSTP = 20
```

Signal: TSTP (keyboard stop)

```
SIGTTIN = 21
```

Signal: TTIN (Terminal input, background)

```
SIGTTOU = 22
```

Signal: TTOU (Terminal output, background)

```
SIGUNUSED = 31
```

Signal: Unused

```
SIGURG = 23
```

Signal: URG (Socket urgent condition)

```
SIGUSR1 = 10
```

Signal: USR1 (User-defined signal 1)

```
SIGUSR2 = 12
```

Signal: USR2 (User-defined signal 2)

```
SIGVTALRM = 26
```

Signal: VTALRM (Virtual alarm clock)

```
SIGWINCH = 28
```

Signal: WINCH (Window/Terminal size change)

```
SIGXCPU = 24
```

Signal: XCPU (CPU limit exceeded)

```
SIGXFSZ = 25
```

Signal: XFSZ (File size limit exceeded)

```
SIG_BLOCK = 0
```

Sigprocmask flags: Add signals to the set of blocked signals.

```
SIG_DFL = 0
```

Signal handler: Default signal handler

```
SIG_ERR = -1
```

Signal handler: error

```
SIG_IGN = 1
```

Signal handler: Ignore signal

```
SIG_SETMASK = 2
```

Sigprocmask flags: Set of blocked signals is given.

```
SIG_UNBLOCK = 1
```

Sigprocmask flags: Remove signals from the set set of blocked signals.

```
SI_PAD_SIZE = ( ( 128 / sizeof ( longint ) ) - 3 )
```

Signal information record pad bytes size. Do not use.

```
SS_DISABLE = 2
```

Socket options

```
SS_ONSTACK = 1
```

Socket options

```
STAT_IFBLK = $6000
```

File (stat (984) record) mode: Block device

```
STAT_IFCHR = $2000
```

File (stat (984) record) mode: Character device

```
STAT_IFDIR = $4000
```

File (stat (984) record) mode: Directory

```
STAT_IFIFO = $1000
```

File (stat (984) record) mode: FIFO

```
STAT_IFLNK = $a000
```

File (stat (984) record) mode: Link

```
STAT_IFMT = $f000
```

File (stat (984) record) mode: File type bit mask

```
STAT_IFREG = $8000
```

File (stat (984) record) mode: Regular file

```
STAT_IFSOCK = $c000
```

File (stat (984) record) mode: Socket

```
STAT_IRGRP = STAT_IROTH shl 3
```

File (stat (984) record) mode: Group read permission

```
STAT_IROTH = $4
```

File (stat (984) record) mode: Other read permission

```
STAT_IRUSR = STAT_IROTH shl 6
```

File (stat (984) record) mode: Owner read permission

```
STAT_IRWXG = STAT_IRWXO shl 3
```

File (stat (984) record) mode: Group permission bits mask

```
STAT_IRWXO = $7
```

File (stat (984) record) mode: Other permission bits mask

```
STAT_IRWXU = STAT_IRWXO shl 6
```

File (stat (984) record) mode: Owner permission bits mask

```
STAT_ISGID = $0400
```

File (stat (984) record) mode: GID bit set

```
STAT_ISUID = $0800
```

File (stat (984) record) mode: UID bit set

```
STAT_ISVTX = $0200
```

File (stat (984) record) mode: Sticky bit set

```
STAT_IWGRP = STAT_IWOTH shl 3
```

File (stat (984) record) mode: Group write permission

```
STAT_IWOTH = $2
```

File (stat (984) record) mode: Other write permission

```
STAT_IWUSR = STAT_IWOTH shl 6
```

File (stat (984) record) mode: Owner write permission

```
STAT_IXGRP = STAT_IXOTH shl 3
```

File (stat (984) record) mode: Others execute permission

```
STAT_IXOTH = $1
```

File (stat (984) record) mode: Others execute permission

```
STAT_IXUSR = STAT_IXOTH shl 6
```

File (stat (984) record) mode: Others execute permission

```
syscall_nr_access = 33
```

```
syscall_nr_acct = 51
```

```
syscall_nr_adjtimex = 124
```

```
syscall_nr_afs_syscall = 137
```

```
syscall_nr_alarm = 27

syscall_nr_bdflush = 134

syscall_nr_break = 17

syscall_nr_brk = 45

syscall_nr_chdir = 12

syscall_nr_chmod = 15

syscall_nr_chown = 16

syscall_nr_chroot = 61

syscall_nr_clone = 120

syscall_nr_close = 6

syscall_nr_creat = 8

syscall_nr_create_module = 127

syscall_nr_delete_module = 129

syscall_nr_dup = 41

syscall_nr_dup2 = 63

syscall_nr_execve = 11

syscall_nr_exit = 1

syscall_nr_fchdir = 133

syscall_nr_fchmod = 94
```

```
syscall_nr_fchown = 95

syscall_nr_fcntl = 55

syscall_nr_fdatasync = 148

syscall_nr_flock = 143

syscall_nr_fork = 2

syscall_nr_fstat = 108

syscall_nr_fstatfs = 100

syscall_nr_fsync = 118

syscall_nr_ftime = 35

syscall_nr_ftruncate = 93

syscall_nr_getdents = 141

syscall_nr_getegid = 50

syscall_nr_geteuid = 49

syscall_nr_getgid = 47

syscall_nr_getgroups = 80

syscall_nr_getitimer = 105

syscall_nr_getpgid = 132

syscall_nr_getpgrp = 65

syscall_nr_getpid = 20
```

```
syscall_nr_getppid = 64

syscall_nr_getpriority = 96

syscall_nr_getresuid = 165

syscall_nr_getrlimit = 76

syscall_nr_getrusage = 77

syscall_nr_getsid = 147

syscall_nr_gettimeofday = 78

syscall_nr_getuid = 24

syscall_nr_get_kernel_syms = 130

syscall_nr_gtty = 32

syscall_nr_idle = 112

syscall_nr_init_module = 128

syscall_nr_ioctl = 54

syscall_nr_ioperm = 101

syscall_nr_iopl = 110

syscall_nr_ipc = 117

syscall_nr_kill = 37

syscall_nr_link = 9

syscall_nr_lock = 53
```

```
syscall_nr_lseek = 19

syscall_nr_lstat = 107

syscall_nr_mkdir = 39

syscall_nr_mknod = 14

syscall_nr_mlock = 150

syscall_nr_mlockall = 152

syscall_nr_mmap = 90

syscall_nr_modify_ldt = 123

syscall_nr_mount = 21

syscall_nr_mprotect = 125

syscall_nr_mpx = 56

syscall_nr_mremap = 163

syscall_nr_msync = 144

syscall_nr_munlock = 151

syscall_nr_munlockall = 153

syscall_nr_munmap = 91

syscall_nr_nanosleep = 162

syscall_nr_nice = 34

syscall_nr_oldfstat = 28
```

```
syscall_nr_oldlstat = 84

syscall_nr_oldolduname = 59

syscall_nr_oldstat = 18

syscall_nr_olduname = 109

syscall_nr_open = 5

syscall_nr_pause = 29

syscall_nr_personality = 136

syscall_nr_phys = 52

syscall_nr_pipe = 42

syscall_nr_poll = 168

syscall_nr_prof = 44

syscall_nr_profil = 98

syscall_nr_ptrace = 26

syscall_nr_query_module = 167

syscall_nr_quotactl = 131

syscall_nr_read = 3

syscall_nr_readdir = 89

syscall_nr_readlink = 85

syscall_nr_readv = 145
```

```
syscall_nr_reboot = 88

syscall_nr_rename = 38

syscall_nr_rmdir = 40

syscall_nr_sched_getparam = 155

syscall_nr_sched_getscheduler = 157

syscall_nr_sched_get_priority_max = 159

syscall_nr_sched_get_priority_min = 160

syscall_nr_sched_rr_get_interval = 161

syscall_nr_sched_setparam = 154

syscall_nr_sched_setscheduler = 156

syscall_nr_sched_yield = 158

syscall_nr_select = 82

syscall_nr_setdomainname = 121

syscall_nr_setfsgid = 139

syscall_nr_setfsuid = 138

syscall_nr_setgid = 46

syscall_nr_setgroups = 81

syscall_nr_sethostname = 74

syscall_nr_setitimer = 104
```

```
syscall_nr_setpgid = 57

syscall_nr_setpriority = 97

syscall_nr_setregid = 71

syscall_nr_setresuid = 164

syscall_nr_setreuid = 70

syscall_nr_setrlimit = 75

syscall_nr_setsid = 66

syscall_nr_settimeofday = 79

syscall_nr_setuid = 23

syscall_nr_setup = 0

syscall_nr_sgetmask = 68

syscall_nr_sigaction = 67

syscall_nr_sigaltstack = 186

syscall_nr_signal = 48

syscall_nr_sigpending = 73

syscall_nr_sigprocmask = 126

syscall_nr_sigreturn = 119

syscall_nr_sigsuspend = 72

syscall_nr_socketcall = 102
```

```
syscall_nr_ssetmask = 69

syscall_nr_stat = 106

syscall_nr_statfs = 99

syscall_nr_stime = 25

syscall_nr_stty = 31

syscall_nr_swapoff = 115

syscall_nr_swapon = 87

syscall_nr_symlink = 83

syscall_nr_sync = 36

syscall_nr_sysfs = 135

syscall_nr_sysinfo = 116

syscall_nr_syslog = 103

syscall_nr_time = 13

syscall_nr_times = 43

syscall_nr_truncate = 92

syscall_nr_ulimit = 58

syscall_nr_umask = 60

syscall_nr_umount = 22

syscall_nr_uname = 122
```

```
syscall_nr_unlink = 10

syscall_nr_uselib = 86

syscall_nr_ustat = 62

syscall_nr_utime = 30

syscall_nr_vhangup = 111

syscall_nr_vm86 = 166

syscall_nr_vm86old = 113

syscall_nr_wait4 = 114

syscall_nr_waitpid = 7

syscall_nr_write = 4

syscall_nr_writev = 146

syscall_nr__llseek = 140

syscall_nr__newselect = 142

syscall_nr__sysctl = 149

Sys_E2BIG = 7

Sys_EACCES = 13

Sys_EADDRINUSE = 98

Sys_EADDRNOTAVAIL = 99

Sys_EADV = 68
```

```
Sys_EAFNOSUPPORT = 97

Sys_EAGAIN = 11

Sys_EALREADY = 114

Sys_EBADE = 52

Sys_EBADF = 9

Sys_EBADFD = 77

Sys_EBADMSG = 74

Sys_EBADR = 53

Sys_EBADRQC = 56

Sys_EBADSLT = 57

Sys_EBFONT = 59

Sys_EBUSY = 16

Sys_ECHILD = 10

Sys_ECHRNG = 44

Sys_ECOMM = 70

Sys_ECONNABORTED = 103

Sys_ECONNREFUSED = 111

Sys_ECONNRESET = 104

Sys_EDEADLK = 35
```

```
Sys_EDEADLOCK = 58

Sys_EDESTADDRREQ = 89

Sys_EDOM = 33

Sys_EDOTDOT = 73

Sys_EDQUOT = 122

Sys_EEXIST = 17

Sys_EFAULT = 14

Sys_EFBIG = 27

Sys_EHOSTDOWN = 112

Sys_EHOSTUNREACH = 113

Sys_EIDRM = 43

Sys_EILSEQ = 84

Sys_EINPROGRESS = 115

Sys_EINTR = 4

Sys_EINVAL = 22

Sys_EIO = 5

Sys_EISCONN = 106

Sys_EISDIR = 21

Sys_EISNAM = 120
```

```
Sys_EL2HLT = 51

Sys_EL2NSYNC = 45

Sys_EL3HLT = 46

Sys_EL3RST = 47

Sys_ELIBACC = 79

Sys_ELIBBAD = 80

Sys_ELIBEXEC = 83

Sys_ELIBMAX = 82

Sys_ELIBSCN = 81

Sys_ELNRNG = 48

Sys_ELOOP = 40

Sys_EMFILE = 24

Sys_EMLINK = 31

Sys_EMSGSIZE = 90

Sys_EMULTIHOP = 72

Sys_ENAMETOOLONG = 36

Sys_ENAVAIL = 119

Sys_ENETDOWN = 100

Sys_ENETRESET = 102
```

```
Sys_ENETUNREACH = 101

Sys_ENFILE = 23

Sys_ENOANO = 55

Sys_ENOBUFS = 105

Sys_ENOCSI = 50

Sys_ENODATA = 61

Sys_ENODEV = 19

Sys_ENOENT = 2

Sys_ENOEXEC = 8

Sys_ENOLCK = 37

Sys_ENOLINK = 67

Sys_ENOMEM = 12

Sys_ENOMSG = 42

Sys_ENONET = 64

Sys_ENOPKG = 65

Sys_ENOPROTOOPT = 92

Sys_ENOSPC = 28

Sys_ENOSR = 63

Sys_ENOSTR = 60
```

```
Sys_ENOSYS = 38

Sys_ENOTBLK = 15

Sys_ENOTCONN = 107

Sys_ENOTDIR = 20

Sys_ENOTEMPTY = 39

Sys_ENOTNAM = 118

Sys_ENOTSOCK = 88

Sys_ENOTTY = 25

Sys_ENOTUNIQ = 76

Sys_ENXIO = 6

Sys_EOPNOTSUPP = 95

Sys_EOVERFLOW = 75

Sys_EPERM = 1

Sys_EPFNOSUPPORT = 96

Sys_EPIPE = 32

Sys_EPROTO = 71

Sys_EPROTONOSUPPORT = 93

Sys_EPROTOTYPE = 91

Sys_ERANGE = 34
```

```
Sys_EREMCHG = 78

Sys_EREMOTE = 66

Sys_EREMOTEIO = 121

Sys_ERESTART = 85

Sys_EROFS = 30

Sys_ERROR_MAX = $fff

Sys_ESHUTDOWN = 108

Sys_ESOCKTNOSUPPORT = 94

Sys_ESPIPE = 29

Sys_ESRCH = 3

Sys_ESRMNT = 69

Sys_ESTALE = 116

Sys_ESTRPIPE = 86

Sys_ETIME = 62

Sys_ETIMEDOUT = 110

Sys_ETOOMANYREFS = 109

Sys_ETXTBSY = 26

Sys_EUCLEAN = 117

Sys_EUNATCH = 49
```

```
Sys_EUSERS = 87

Sys_EWOULDBLOCK = Sys_EAGAIN

Sys_EXDEV = 18

Sys_EXFULL = 54

TAB0 = $0000000

TAB1 = $0000800

TAB2 = $0001000

TAB3 = $0001800

TABDLY = $0001800

TCFLSH = $540B

TCGETA = $5405

TCGETS = $5401

TCIFLUSH = 0

TCIOFF = 2

TCIOFLUSH = 2

TCION = 3

TCOFLUSH = 1

TCOOFF = 0

TCOON = 1
```

```
TCSADRAIN = 1

TCSAFLUSH = 2

TCSANOW = 0

TCSBRK = $5409

TCSBRKP = $5425

TCSETA = $5406

TCSETAF = $5408

TCSETAW = $5407

TCSETS = $5402

TCSETSF = $5404

TCSETSW = $5403

TCXONC = $540A

TIOCCONS = $541D

TIOCEXCL = $540C

TIOCGETD = $5424

TIOCGICOUNT = $545D

TIOCGLCKTRMIOS = $5456

TIOCGPGRP = $540F

TIOCGSERIAL = $541E
```

```
TIOCGSOFTCAR = $5419

TIOCGWINSZ = $5413

TIOCINQ = FIONREAD

TIOCLINUX = $541C

TIOCMBIC = $5417

TIOCMBIS = $5416

TIOCMGET = $5415

TIOCMIWAIT = $545C

TIOCMSET = $5418

TIOCM_CAR = $040

TIOCM_CD = TIOCM_CAR

TIOCM_CTS = $020

TIOCM_DSR = $100

TIOCM_DTR = $002

TIOCM_LE = $001

TIOCM_OUT1 = $2000

TIOCM_OUT2 = $4000

TIOCM_RI = TIOCM_RNG

TIOCM_RNG = $080
```

```
TIOCM_RTS = $004

TIOCM_SR = $010

TIOCM_ST = $008

TIOCNOTTY = $5422

TIOCNXCL = $540D

TIOCOUTQ = $5411

TIOCPKT = $5420

TIOCPKT_DATA = 0

TIOCPKT_DOSTOP = 32

TIOCPKT_FLUSHREAD = 1

TIOCPKT_FLUSHWRITE = 2

TIOCPKT_NOSTOP = 16

TIOCPKT_START = 8

TIOCPKT_STOP = 4

TIOCSCTTY = $540E

TIOCSERCONFIG = $5453

TIOCSERGETLSR = $5459

TIOCSERGETMULTI = $545A

TIOCSERGSTRUCT = $5458
```

```
TIOCSERGWILD = $5454

TIOCSERSETMULTI = $545B

TIOCSERSWILD = $5455

TIOCSETD = $5423

TIOCSLCKTRMIOS = $5457

TIOCSPGRP = $5410

TIOCSSERIAL = $541F

TIOCSSOFTCAR = $541A

TIOCSTI = $5412

TIOCSWINSZ = $5414

TIOCTTYGSTRUCT = $5426

TOSTOP = $0000100

VDISCARD = 13

VEOF = 4

VEOL = 11

VEOL2 = 16

VERASE = 2

VINTR = 0

VKILL = 3
```

```
VLNEXT = 15
```

```
VMIN = 6
```

```
VQUIT = 1
```

```
VREPRINT = 12
```

```
VSTART = 8
```

```
VSTOP = 9
```

```
VSUSP = 10
```

```
VSWTC = 7
```

```
VT0 = $0000000
```

```
VT1 = $0004000
```

```
VTDLY = $0004000
```

```
VTIME = 5
```

```
VWERASE = 14
```

```
Wait_Any = -1
```

WaitPID (1063): Wait on any process

```
Wait_Clone = $80000000
```

WaitPID (1063): Wait on clone processes only.

```
Wait_MyPGRP = 0
```

WaitPID (1063): Wait processes from current process group

```
Wait_NoHang = 1
```

WaitPID (1063): Do not wait

```
Wait_UnTraced = 2
```

WaitPID (1063): Also report stopped but untraced processes

```
WNOHANG = $1
```

Waitpid (1063) option: Do not wait for processes to terminate.

```
WUNTRACED = $2
```

Waitpid (1063) option: Also report children wich were stopped but not yet reported

```
W_OK = 2
```

Access (989) call test: write allowed

```
XCASE = $0000004
```

```
XTABS = $0001800
```

```
X_OK = 1
```

Access (989) call test: execute allowed

```
__WCLONE = $80000000
```

Waitpid option: Wait for clone children only

23.11.2 Types

```
ComStr =
```

Command-line string type.

```
dev_t = Word
```

Device descriptor type

```
dirent = packed record
  ino : LongInt;
  off : LongInt;
  reclen : Word;
  name : Array[0..255] of Char;
end
```

Record used in the ReadDir (1041) function to return files in a directory.

```
DirStr =
```

Filename directory part string type.

```
ExtStr =
```

Filename extension part string type.

```
fdSet = Array[0..7] of LongInt
```

Array containing file descriptor bitmask for the Select (1043) call.

```
NameStr =
```

Filename name part string type.

```
PathStr =
```

Filename path part string type.

```
PDir = ^TDir
```

Pointer to TDir (985) record

```
pdirent = ^dirent
```

Pointer to Dirent (980) record.

```
pfdset = ^fdSet
```

Pointer to FDSet (1013) array.

```
pfpstate = ^tfpstate
```

Pointer to tfpstate (986) record.

```
pglob = ^tglob
```

Pointer to TGlob (986) record.

```
PSigActionRec = ^SigActionRec
```

Pointer to SigActionRec (983) record.

```
PSigAltStack = ^SigAltStack
```

Pointer to SigAltStack (983) record

```
PSigContextRec = ^SigContextRec
```

Pointer to SigContextRec (983) record

```
PSignalHandler = ^SignalHandler
```

Pointer to SignalHandler (983) type.

```
PSignalRestorer = ^SignalRestorer
```

Pointer to SignalRestorer (983) type

```
PSigSet = ^SigSet
```

Pointer to signal set.

```
pstack_t = ^stack_t
```

Pointer to stack_t (984) record

```
PStat = ^Stat
```

Pointer to Stat (984) record.

```
PStatFS = ^Statfs
```

Pointer to StatFS (984) record.

```
PSysCallRegs = ^SysCallRegs
```

Pointer to SysCallRegs (985) record.

```
PSysInfo = ^TSysinfo
```

Pointer to TSysInfo (988) record.

```
ptimeval = ^timeval
```

Pointer to TTimeVal (988) record

```
ptimezone = ^timezone
```

Pointer to TimeZone (987) record.

```
PUTimeBuf = ^UTimeBuf
```

Pointer to UTimeBuf (988) record

```
PUTSName = ^utsname
```

Pointer to UTSName (989) record.

```
SigActionRec = packed record
  Handler : record
  end;
  Sa_Mask : SigSet;
  Sa_Flags : LongInt;
  Sa_restorer : SignalRestorer;
end
```

Record used in SigAction (1047) call.

```
SigAltStack = record
  ss_sp : pointer;
  ss_flags : LongInt;
  ss_size : Size_T;
end
```

Alternate stack registers record

```
SigContextRec = record
  gs : Word;
  __gsh : Word;
  fs : Word;
  __fsh : Word;
  es : Word;
  __esh : Word;
  ds : Word;
  __dsh : Word;
  edi : cardinal;
  esi : cardinal;
  ebp : cardinal;
  esp : cardinal;
  ebx : cardinal;
  edx : cardinal;
  ecx : cardinal;
  eax : cardinal;
  trapno : cardinal;
  err : cardinal;
  eip : cardinal;
  cs : Word;
  __csh : Word;
  eflags : cardinal;
  esp_at_signal : cardinal;
  ss : Word;
  __ssh : Word;
  fpstate : pfpstate;
  oldmask : cardinal;
  cr2 : cardinal;
end
```

The above records contain information about the processor state and process state at the moment a signal is sent to your program.

```
SignalHandler = procedure(Sig: LongInt)
```

Function prototype for the Signal (1048) call.

```
SignalRestorer = procedure
```

Signal restorer function prototype

```
SigSet = LongInt
```

Signal set type

```
Size_T = cardinal
```

Size type

```
stack_t = SigAltStack
```

Alias for SigAltStack (983) type

```
Stat = packed record
  dev : dev_t;
  pad1 : Word;
  ino : LongInt;
  mode : Word;
  nlink : Word;
  uid : Word;
  gid : Word;
  rdev : dev_t;
  pad2 : Word;
  size : LongInt;
  blksize : LongInt;
  blocks : LongInt;
  atime : LongInt;
  unused1 : LongInt;
  mtime : LongInt;
  unused2 : LongInt;
  ctime : LongInt;
  unused3 : LongInt;
  unused4 : LongInt;
  unused5 : LongInt;
end
```

Record describing an inode (file) in the fstat (1019) call.

```
Statfs = packed 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
```

Record describing a file system in the fsstat (1018) call.

```
SysCallRegs = record
  reg1 : LongInt;
  reg2 : LongInt;
  reg3 : LongInt;
  reg4 : LongInt;
  reg5 : LongInt;
  reg6 : LongInt;
end
```

Register describing system calls.

```
TCloneFunc = function(args: pointer) : LongInt
```

Clone function prototype.

```
TDir = packed record
  fd : Integer;
  loc : LongInt;
  size : Integer;
  buf : pdirent;
  nextoff : LongInt;
  dd_max : Integer;
  lock : pointer;
end
```

Record used in OpenDir (1039) and ReadDir (1041) calls

```
TDirEnt = dirent
```

Alias for DirEnt (980) record

```
Termio = packed record
  c_iflag : Word;
  c_oflag : Word;
  c_cflag : Word;
  c_lflag : Word;
  c_line : Word;
  c_cc : Array[0..NCC-1] of Char;
end
```

Terminal I/O description record (small)

```
Termios = record
  c_iflag : Cardinal;
  c_oflag : Cardinal;
  c_cflag : Cardinal;
  c_lflag : Cardinal;
  c_line : Char;
  c_cc : Array[0..NCCS-1] of Byte;
```

```
  c_ispeed : LongInt;
  c_ospeed : LongInt;
end
```

Terminal I/O description record

```
TFDSet = fdSet
```

Alias for FDSet (1013) type.

```
tfpreg = record
  significand : Array[0..3] of Word;
  exponent : Word;
end
```

Record describing floating point register in signal handler.

```
tfpstate = record
  cw : cardinal;
  sw : cardinal;
  tag : cardinal;
  ipoff : cardinal;
  cssel : cardinal;
  dataoff : cardinal;
  datasel : cardinal;
  st : Array[0..7] of tfpreg;
  status : cardinal;
end
```

Record describing floating point unit in signal handler.

```
tglob = record
  name : pchar;
  next : pglob;
end
```

Record containing one entry in the result of Glob (1028)

```
timespec = packed record
  tv_sec : LongInt;
  tv_nsec : LongInt;
end
```

Time interval for the NanoSleep (1037) function.

```
timeval = packed record
  sec : LongInt;
  usec : LongInt;
end
```

Record specifying a time inteval.

```
timezone = packed record
  minuteswest : LongInt;
  dsttime : LongInt;
end
```

Record describing a timezone

```
tmmapargs = record
  address : LongInt;
  size : LongInt;
  prot : LongInt;
  flags : LongInt;
  fd : LongInt;
  offset : LongInt;
end
```

Record containing mmap args.

```
Tpipe = Array[1..2] of LongInt
```

Array describing a pipe pair of filedescriptors.

```
TSigAction = procedure(Sig: LongInt;SigContext: SigContextRec)
```

Function prototype for SigAction (1047) call.

```
TStat = Stat
```

Alias for Stat (984) record.

```
TStatFS = Statfs
```

Alias for StatFS (984) type.

```
TSysCallRegs = SysCallRegs
```

Alias for SysCallRegs (985) record

```
TSysinfo = packed record
  uptime : LongInt;
  loads : Array[1..3] of LongInt;
  totalram : LongInt;
  freeram : LongInt;
  sharedram : LongInt;
  bufferram : LongInt;
  totalswap : LongInt;
  freeswap : LongInt;
  procs : Integer;
  s : String;
end
```

Record with system information, used by the SysInfo (1053) call.

```
TTermio = Termio
```

Alias for TermIO (985) record

```
TTermios = Termios
```

Alias for Termios (986) record.

```
TTimeVal = timeval
```

Alias for TimeVal (987) record.

```
TTimeZone = timezone
```

Alias for TimeZone (987) record.

```
TUTimeBuf = UTimeBuf
```

Alias for UTimBuf (988) record.

```
TUTSName = utsname
```

Alias for UTSName (989) record.

```
TWinSize = winsize
```

Alias for WinSize (989) record.

```
UTimBuf = packed record
  actime : LongInt;
  modtime : LongInt;
end
```

Record used in Utime (1062) to set file access and modificaton times.

```
UTimeBuf = UTimBuf
```

Alias for UTimBuf (988) record.

```
utsname = packed record
  sysname : Array[0..64] of Char;
  nodename : Array[0..64] of Char;
  release : Array[0..64] of Char;
  version : Array[0..64] of Char;
  machine : Array[0..64] of Char;
  domainname : Array[0..64] of Char;
end
```

The elements of this record are null-terminated C style strings, you cannot access them directly.

```
winsize = packed record
  ws_row : Word;
  ws_col : Word;
  ws_xpixel : Word;
  ws_ypixel : Word;
end
```

Record describing terminal window size.

23.11.3 Variables

`ErrNo : LongInt`

Error number of last operation.

`LinuxError : LongInt`

`Linuxerror` is the variable in which the procedures in the linux unit report errors.

`tzdaylight : Boolean`

Indicates whether daylight savings time is active.

`tzname : Array[boolean] of pchar`

Timezone name.

`tzseconds : LongInt`

Seconds west of GMT

23.12 Procedures and functions

23.12.1 Access

Synopsis: Check file access

Declaration: `function Access(Path: PathStr;mode: Integer) : Boolean`

Visibility: default

Description: `Access` tests user's access rights on the specified file. `Mode` is a mask existing of one or more of the following:

R_OK User has read rights.

W_OK User has write rights.

X_OK User has execute rights.

F_OK File exists.

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_enoent A directory component in `Path` doesn't exist or is a dangling symbolic link.

sys_enotdir A directory component in `Path` is not a directory.

sys_enomem Insufficient kernel memory.

sys_eloop `Path` has a circular symbolic link.

See also: Chown (996), Chmod (995)

Listing: ./olinuxex/ex26.pp

```
Program Example26;

{ Program to demonstrate the Access function. }

Uses oldlinux;

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.
```

23.12.2 Alarm

Synopsis: Schedule an alarm signal to be delivered

Declaration: `function Alarm(Sec: LongInt) : LongInt`

Visibility: default

Description: `Alarm` schedules an alarm signal to be delivered to your process in `Sec` seconds. When `Sec` seconds have elapsed, Linux will send a `SIGALRM` signal to the current process. If `Sec` is zero, then no new alarm will be set. Whatever the value of `Sec`, any previous alarm is cancelled.

The function returns the number of seconds till the previously scheduled alarm was due to be delivered, or zero if there was none.

See also: SigAction (1047)

Listing: ./olinuxex/ex59.pp

```
Program Example59;

{ Program to demonstrate the Alarm function. }

Uses oldlinux;

Procedure AlarmHandler(Sig : longint);cdecl;
```

```
begin
  Writeln ('Got to alarm handler');
end;

begin
  Writeln('Setting alarm handler');
  Signal(SIGALRM,@AlarmHandler);
  Writeln ('Scheduling Alarm in 10 seconds');
  Alarm(10);
  Writeln ('Pausing');
  Pause;
  Writeln ('Pause returned');
end.
```

23.12.3 AssignPipe

Synopsis: Create a set of pipe file handlers

Declaration:
```
function AssignPipe(var pipe_in: LongInt;var pipe_out: LongInt)
                      : Boolean
function AssignPipe(var pipe_in: text;var pipe_out: text) : Boolean
function AssignPipe(var pipe_in: File;var pipe_out: File) : Boolean
```

Visibility: default

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`.

This call is overloaded. The in and out pipe can take three forms: an typed or untyped file, a text file or a file descriptor.

If a text file is passed then reading and writing from/to the pipe can be done through the usual `Readln(Pipe_in,...)` and `Writeln(Pipe_out,...)` procedures.

The function returns `True` if everything went succesfully, `False` otherwise.

Errors: In case the function fails and returns `False`, `LinuxError` is used to report errors:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: POpen (1040), MkFifo (1035)

Listing: ./olinuxex/ex36.pp

```
Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses oldlinux;

Var pipi,pipo : Text;
    s : String;

begin
  Writeln ('Assigning Pipes.');
  If Not assignpipe(pipi,pipo) then
```

```
    Writeln('Error assigning pipes !',LinuxError);
  Writeln ('Writing to pipe, and flushing.');
  Writeln (pipo,'This is a textstring');close(pipo);
  Writeln ('Reading from pipe.');
  While not eof(pipi) do
    begin
    Readln (pipi,s);
    Writeln ('Read from pipe : ',s);
    end;
  close (pipi);
  writeln ('Closed pipes.');
  writeln
end.
```

23.12.4 AssignStream

Synopsis: Assign stream for in and output to a program

Declaration:
```
function AssignStream(var StreamIn: text;var Streamout: text;
                      const Prog: String) : LongInt
function AssignStream(var StreamIn: Text;var StreamOut: Text;
                      var StreamErr: Text;const prog: String) : LongInt
```

Visibility: default

Description: `AssignStream` creates a 2 or 3 pipes, i.e. two (or three) file objects, one for input, one for output,(and one for standard error) the other ends of these pipes are connected to standard input and output (and standard error) of `Prog`. `Prog` is the name of a program (including path) with options, which will be executed.

What is written to `StreamOut`, will go to the standard input of `Prog`. Whatever is written by `Prog` to it's standard output can be read from `StreamIn`. Whatever is written by `Prog` to it's standard error read from `StreamErr`, if present.

Reading and writing happens through the usual `Readln(StreamIn,...)` and `Writeln (StreamOut,...)` procedures.

Remark: You should *not* use `Reset` or `Rewrite` on a file opened with `POpen`. This will close the file before re-opening it again, thereby closing the connection with the program.

The function returns the process ID of the spawned process, or -1 in case of error.

Errors: In case of error (return value -1) `LinuxError` is used to report errors:

sys_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 (991), POpen (1040)

Listing: ./olinuxex/ex38.pp

```
Program Example38;

{ Program to demonstrate the AssignStream function. }

Uses oldlinux;
```

```
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.
```

23.12.5 Basename

Synopsis: Return basename of a file

Declaration: `function Basename(const path: PathStr;const suf: PathStr) : PathStr`

Visibility: default

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 (1000), FExpand (1014)

Listing: ./olinuxex/ex48.pp

```
Program Example48;

{ Program to demonstrate the BaseName function. }

Uses oldlinux;

Var S : String;

begin
  S:=FExpand(Paramstr(0));
  Writeln ('This program is called : ',Basename(S,''));
end.
```

23.12.6 CFMakeRaw

Synopsis: Sets flags in Termios (986) record.

Declaration: `procedure CFMakeRaw(var tios: Termios)`

Visibility: default

Description: `CFMakeRaw` sets the flags in the `Termios` structure `Tios` to a state so that the terminal will function in Raw Mode.

For an example, see TCGetAttr (1058).

Errors: None.

See also: CFSetOSpeed (994), CFSetISpeed (994)

23.12.7 CFSetISpeed

Synopsis: Set input baud rate in Termios (986) record

Declaration: `procedure CFSetISpeed(var tios: Termios;speed: Cardinal)`

Visibility: default

Description: `CFSetISpeed` Sets the input baudrate in the `TermIOS` structure `Tios` to `Speed`.

Errors: None.

See also: CFSetOSpeed (994), CFMakeRaw (994)

23.12.8 CFSetOSpeed

Synopsis: Set output baud rate in Termios (986) record

Declaration: `procedure CFSetOSpeed(var tios: Termios;speed: Cardinal)`

Visibility: default

Description: `CFSetOSpeed` Sets the output baudrate in the `Termios` structure `Tios` to `Speed`.

Errors: None.

See also: CFSetISpeed (994), CFMakeRaw (994)

23.12.9 Chmod

Synopsis: Change file permission bits

Declaration: `function Chmod(path: PathStr;Newmode: LongInt) : Boolean`

Visibility: default

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_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 (996), Access (989), Octal (1038)

Listing: ./olinuxex/ex23.pp

```
Program Example23;

{ Program to demonstrate the Chmod function. }

Uses oldlinux;

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.
```

23.12.10 Chown

Synopsis: Change owner of file

Declaration: `function Chown(path: PathStr;NewUid: LongInt;NewGid: LongInt) : Boolean`

Visibility: default

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_enoent A directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent 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 (995), Access (989)

Listing: ./olinuxex/ex24.pp

```
Program Example24;

{ Program to demonstrate the Chown function. }

Uses oldlinux;
```

```
Var UID,GID : Longint;
    F : Text;

begin

  WriteIn ('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);
  WriteIn (f,'The owner of this file should become : ');
  WriteIn (f,'UID : ',UID);
  WriteIn (f,'GID : ',GID);
  Close (F);
  if not Chown ('test.txt',UID,GID) then
    if LinuxError=Sys_EPERM then
      WriteIn ('You are not root !')
    else
      WriteIn ('Chmod failed with exit code : ',LinuxError)
  else
    WriteIn ('Changed owner successfully !');
end.
```

23.12.11 Clone

Synopsis: Clone current process (create new thread)

Declaration:
```
function Clone(func: TCloneFunc;sp: pointer;flags: LongInt;
                  args: pointer) : LongInt
```

Visibility: default

Description: `Clone` creates a child process which is a copy of the parent process, just like Fork (1016) does. In difference with `Fork`, however, the child process shares some parts of it's execution context with its parent, so it is suitable for the implementation of threads: many instances of a program that share the same memory.

When the child process is created, it starts executing the function `Func`, and passes it `Args`. The return value of `Func` is either the explicit return value of the function, or the exit code of the child process.

The `sp` pointer points to the memory reserved as stack space for the child process. This address should be the top of the memory block to be used as stack.

The `Flags` determine the behaviour of the `Clone` call. The low byte of the Flags contains the number of the signal that will be sent to the parent when the child dies. This may be bitwise OR'ed with the following constants:

CLONE_VM Parent and child share the same memory space, including memory (un)mapped with subsequent `mmap` calls.

CLONE_FS Parent and child have the same view of the filesystem; the `chroot`, `chdir` and `umask` calls affect both processes.

CLONE_FILES the file descriptor table of parent and child is shared.

CLONE_SIGHAND the parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PIDPArent and child have the same process ID.

Clone returns the process ID in the parent process, and -1 if an error occurred.

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Too many processes are running.

sys_enomem Not enough memory to create child process.

See also: Fork (1016)

Listing: ./olinuxex/ex71.pp

```
program TestC{lone};

uses
  oldlinux, Errors, crt;

const
  Ready : Boolean = false;
  aChar : Char    = 'a';

function CloneProc( Arg: Pointer ): LongInt; Cdecl;
begin
  WriteLn('Hello from the clone ',PChar(Arg));
  repeat
    Write(aChar);
    Select(0,Nil,Nil,Nil,Nil);
  until Ready;
  WriteLn( 'Clone finished.');
  CloneProc := 1;
end;

var
  PID : LongInt;

procedure MainProc;
begin
  WriteLn('cloned process PID: ', PID );
  WriteLn('Press <ESC> to kill ... ' );
  repeat
    Write('.');
    Select(0,Nil,Nil,Nil,Nil);
    if KeyPressed then
      case ReadKey of
        #27: Ready := true;
        'a': aChar := 'A';
        'A': aChar := 'a';
        'b': aChar := 'b';
        'B': aChar := 'B';
      end;
  until Ready;
  WriteLn('Ready.');
end;

const
  StackSze = 16384;
  theFlags = CLONE_VM+CLONE_FS+CLONE_FILES+CLONE_SIGHAND;
```

```
  aMsg      : PChar = 'Oops !';

var
  theStack : Pointer;
  ExitStat : LongInt;

begin
  GetMem(theStack,StackSze);
  PID := Clone(@CloneProc,
               Pointer( LongInt(theStack)+StackSze),
               theFlags,
               aMsg);
  if PID < 0 then
    WriteLn('Error : ', LinuxError, ' when cloning.')
  else
    begin
    MainProc;
    case WaitPID(0,@ExitStat,Wait_Untraced or wait_clone) of
      -1: WriteLn('error:',LinuxError,'; ',StrError(LinuxError));
       0: WriteLn('error:',LinuxError,'; ',StrError(LinuxError));
    else
      WriteLn('Clone exited with: ',ExitStat shr 8);
    end;
    end;
  FreeMem( theStack, StackSze );
end.
```

23.12.12 CloseDir

Synopsis: Close directory file descriptor

Declaration: `function CloseDir(p: PDir) : Integer`

Visibility: default

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

For an example, see OpenDir (1039).

Errors: Errors are returned in LinuxError.

See also: OpenDir (1039), ReadDir (1041), SeekDir (1043), TellDir (1060)

23.12.13 CreateShellArgV

Synopsis: Create an array of null-terminated strings

Declaration: `function CreateShellArgV(const prog: String) : ppchar`
`function CreateShellArgV(const prog: Ansistring) : ppchar`

Visibility: default

Description: `CreateShellArgV` creates an array of 3 `PChar` pointers that can be used as arguments to `ExecVE` the first elements in the array will contain `/bin/sh`, the second will contain `-c`, and the third will contain `prog`.

The function returns a pointer to this array, of type `PPChar`.

Errors: None.

See also: Shell (1046)

Listing: ./olinuxex/ex61.pp

```
Program ex61;

{ Example program to demonstrate the CreateShellArgV function }

uses oldlinux;

Var
  S: String;
  PP : PPchar;
   I : longint;

begin
  S:='script -a -b -c -d -e fghijk';
  PP:=CreateShellArgV(S);
  I:=0;
  If PP<>Nil then
    While PP[i]<>Nil do
      begin
      Writeln ('Got : "',PP[i],'"');
      Inc(i);
      end;
end.
```

23.12.14 Dirname

Synopsis: Extract directory part from filename

Declaration: `function Dirname(const path: PathStr) : PathStr`

Visibility: default

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 (993), FExpand (1014)

Listing: ./olinuxex/ex47.pp

```
Program Example47;

{ Program to demonstrate the DirName function. }

Uses oldlinux;

Var S : String;

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

23.12.15 Dup

Synopsis: Duplicate a file handle

Declaration:
```
function Dup(oldfile: LongInt;var newfile: LongInt) : Boolean
function Dup(var oldfile: text;var newfile: text) : Boolean
function Dup(var oldfile: File;var newfile: File) : Boolean
```

Visibility: default

Description: Makes `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile` in case it is a Text file or untyped file. Due to the buffering mechanism of Pascal, this has not the same functionality as the dup call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns `False` in case of an error, `True` if successful.

Errors: In case of errors, `Linuxerror` is used to report errors.

sys_ebadf`OldFile` hasn't been assigned.

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

See also: Dup2 (1001)

Listing: ./olinuxex/ex31.pp

```
program Example31;

{ Program to demonstrate the Dup function. }

uses oldlinux;

var f : text;

begin
  if not dup (output,f) then
    WriteIn ('Dup Failed !');
  writeln ('This is written to stdout.');
  writeln (f,'This is written to the dup file , and flushed');flush(f);
  writeln
end.
```

23.12.16 Dup2

Synopsis: Duplicate one filehandle to another

Declaration:
```
function Dup2(oldfile: LongInt;newfile: LongInt) : Boolean
function Dup2(var oldfile: text;var newfile: text) : Boolean
function Dup2(var oldfile: File;var newfile: File) : Boolean
```

Visibility: default

Description: Makes `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile` in the case of text or untyped files.

`NewFile` can be an assigned file. If `newfile` was open, it is closed first. Due to the buffering mechanism of Pascal, this has not the same functionality as the `dup2` call in C. The internal Pascal

buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns `True` if succesful, false otherwise.

Errors: In case of error, `Linuxerror` is used to report errors.

sys_ebadf`OldFile` hasn't been assigned.

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

See also: Dup (1001)

Listing: ./olinuxex/ex32.pp

```
program Example31;

{ Program to demonstrate the Dup function. }

uses oldlinux;

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
    WriteIn ('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.
```

23.12.17 EpochToLocal

Synopsis: Convert epoch time to local time

Declaration:
```
procedure EpochToLocal(epoch: LongInt;var year: Word;var month: Word;
                       var day: Word;var hour: Word;var minute: Word;
                       var second: Word)
```

Visibility: default

Description: Converts the epoch time (=Number of seconds since 00:00:00 , January 1, 1970, corrected for your time zone) to local date and time.

This function takes into account the timzeone settings of your system.

Errors: None

See also: GetEpochTime (1022), LocalToEpoch (1032), GetTime (1027), GetDate (1020)

Listing: ./olinuxex/ex3.pp

```
Program Example3;

{ Program to demonstrate the EpochToLocal function. }

Uses oldlinux;

Var Year,month,day,hour,minute,seconds : Word;

begin
  EpochToLocal (GetEpochTime,Year,month,day,hour,minute,seconds);
  WriteIn ('Current date : ',Day:2,'/',Month:2,'/',Year:4);
  WriteIn ('Current time : ',Hour:2,':',minute:2,':',seconds:2);
end.
```

23.12.18 Execl

Synopsis: Execute process (using argument list)

Declaration:
```
procedure Execl(const Todo: String)
procedure Execl(const Todo: Ansistring)
```

Visibility: default

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 (1006), Execv (1005), Execvp (1007), Execle (1004), Execlp (1004), Fork (1016)

Listing: ./olinuxex/ex10.pp

```
Program Example10;

{ Program to demonstrate the Execl function. }

Uses oldlinux, strings;

begin
  { Execute 'ls -l', with current environment. }
  { 'ls' is NOT looked for in PATH environment variable.}
  Execl ('/bin/ls -l');
end.
```

23.12.19 Execle

Synopsis: Execute process (using argument list, environment)

Declaration:
```
procedure Execle(Todo: String;Ep: ppchar)
procedure Execle(Todo: AnsiString;Ep: ppchar)
```

Visibility: default

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_enotdir A component of the path is not a directory.

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

See also: Execve (1006), Execv (1005), Execvp (1007), Execl (1003), Execlp (1004), Fork (1016)

Listing: ./olinuxex/ex11.pp

```
Program Example11;

{ Program to demonstrate the Execle function. }

Uses oldlinux, 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.
```

23.12.20 Execlp

Synopsis: Execute process (using argument list, environment; search path)

Declaration:
```
procedure Execlp(Todo: String;Ep: ppchar)
procedure Execlp(Todo: Ansistring;Ep: ppchar)
```

Visibility: default

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 (1006), Execv (1005), Execvp (1007), Execle (1004), Execl (1003), Fork (1016)

Listing: ./olinuxex/ex12.pp

```
Program Example12;

{ Program to demonstrate the Execlp function. }

Uses oldlinux, 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.
```

23.12.21 Execv

Synopsis: Execute process

Declaration:
```
procedure Execv(const path: PathStr;args: ppchar)
procedure Execv(const path: AnsiString;args: ppchar)
```

Visibility: default

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 (1006), Execvp (1007), Execle (1004), Execl (1003), Execlp (1004), Fork (1016)

Listing: ./olinuxex/ex8.pp

```
Program Example8;

{ Program to demonstrate the Execv function. }

Uses oldlinux, 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.
```

23.12.22 Execve

Synopsis: Execute process using environment

Declaration:
```
procedure Execve(Path: PathStr;args: ppchar;ep: ppchar)
procedure Execve(Path: AnsiString;args: ppchar;ep: ppchar)
procedure Execve(path: pchar;args: ppchar;ep: ppchar)
```

Visibility: default

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`:

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

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve (1006), Execv (1005), Execvp (1007), Execle (1004), Execl (1003), Execlp (1004), Fork (1016)

Listing: ./olinuxex/ex7.pp

```
Program Example7;

{ Program to demonstrate the Execve function. }

Uses oldlinux, 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.
```

23.12.23 Execvp

Synopsis: Execute process, search path

Declaration:
```
procedure Execvp(Path: PathStr;Args: ppchar;Ep: ppchar)
procedure Execvp(Path: AnsiString;Args: ppchar;Ep: ppchar)
```

Visibility: default

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 (1006), Execv (1005), Execle (1004), Execl (1003), Execlp (1004), Fork (1016)

Listing: ./olinuxex/ex9.pp

```
Program Example9;

{ Program to demonstrate the Execvp function. }

Uses oldlinux, 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.
```

23.12.24 ExitProcess

Synopsis: Exit the current process

Declaration: `procedure ExitProcess(val: LongInt)`

Visibility: default

Description: `ExitProcess` exits the currently running process, and report `Val` as the exit status.

Remark: If this call is executed, the normal unit finalization code will not be executed. This may lead to unexpected errors and stray files on your system. It is therefore recommended to use the `Halt` call instead.

Errors: None.

See also: Fork (1016), ExecVE (1006)

23.12.25 Fcntl

Synopsis: File control operations.

Declaration:
```
function Fcntl(Fd: LongInt;Cmd: LongInt) : LongInt
procedure Fcntl(Fd: LongInt;Cmd: LongInt;Arg: LongInt)
function Fcntl(var Fd: Text;Cmd: LongInt) : LongInt
procedure Fcntl(var Fd: Text;Cmd: LongInt;Arg: LongInt)
```

Visibility: default

Description: Read a file's attributes. `Fd` is an assigned file, or a valid file descriptor. `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.

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.

23.12.26 fdClose

Synopsis: Close file descriptor

Declaration: `function fdClose(fd: LongInt) : Boolean`

Visibility: default

Description: `fdClose` closes a file with file descriptor `Fd`. The function returns `True` if the file was closed successfully, `False` otherwise.

For an example, see fdOpen (1010).

Errors: Errors are returned in LinuxError.

See also: fdOpen (1010), fdRead (1011), fdWrite (1013), fdTruncate (1012), fdFlush (1009), fdSeek (1012)

23.12.27 fdFlush

Synopsis: Flush kernel file buffer

Declaration: `function fdFlush(fd: LongInt) : Boolean`

Visibility: default

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.

For an example, see fdRead (1011).

Errors: Errors are returned in LinuxError.

See also: fdOpen (1010), fdClose (1009), fdRead (1011), fdWrite (1013), fdTruncate (1012), fdSeek (1012)

23.12.28 fdOpen

Synopsis: Open file and return file descriptor

Declaration:
```
function fdOpen(pathname: String;flags: LongInt) : LongInt
function fdOpen(pathname: String;flags: LongInt;mode: LongInt) : LongInt
function fdOpen(pathname: pchar;flags: LongInt) : LongInt
function fdOpen(pathname: pchar;flags: LongInt;mode: LongInt) : LongInt
```

Visibility: default

Description: `fdOpen` opens a file in `PathName` with flags `flags` One of the following:

Open_RdOnly File is opened Read-only

Open_WrOnly File is opened Write-only

Open_RdWr File is opened Read-Write

The flags may be`OR`-ed with one of the following constants:

Open_Creat File is created if it doesn't exist.

Open_Excl If the file is opened with `Open_Creat` and it already exists, the call wil fail.

Open_NoCtty If the file is a terminal device, it will NOT become the process' controlling terminal.

Open_Trunc If the file exists, it will be truncated.

Open_Appendthe file is opened in append mode. *Before each write*, the file pointer is positioned at the end of the file.

Open_NonBlock The file is opened in non-blocking mode. No operation on the file descriptor will cause the calling process to wait till.

Open_NDelay Idem as `Open_NonBlock`

Open_Sync The file is opened for synchronous IO. Any write operation on the file will not return untill the data is physically written to disk.

Open_NoFollowif the file is a symbolic link, the open fails. (linux 2.1.126 and higher only)

Open_Directoryif the file is not a directory, the open fails. (linux 2.1.126 and higher only)

`PathName` can be of type `PChar` or `String`. The optional `mode` argument specifies the permissions to set when opening the file. This is modified by the umask setting. The real permissions are `Mode and not umask`. The return value of the function is the filedescriptor, or a negative value if there was an error.

Errors: Errors are returned in LinuxError.

See also: fdClose (1009), fdRead (1011), fdWrite (1013), fdTruncate (1012), fdFlush (1009), fdSeek (1012)

Listing: ./olinuxex/ex19.pp

```
Program Example19;

{ Program to demonstrate the fdOpen, fdwrite and fdCLose functions. }

Uses oldlinux;

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.
```

23.12.29 fdRead

Synopsis: Read data from file descriptor

Declaration: `function fdRead(fd: LongInt;var buf;size: LongInt) : LongInt`

Visibility: default

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 (1010), fdClose (1009), fdWrite (1013), fdTruncate (1012), fdFlush (1009), fdSeek (1012)

Listing: ./olinuxex/ex20.pp

```
Program Example20;

{ Program to demonstrate the fdRead and fdTruncate functions. }

Uses oldlinux;

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

Var FD : Longint;
    I : longint;

begin
  FD:=fdOpen('test.dat',open_wronly or open_creat,octal(666));
  if fd>0 then
    begin
    { Fill file with data }
    for I:=1 to 10 do
      if fdWrite (FD,Data[1],10)<>10 then
        begin
        writeln ('Error when writing !');
        halt(1);
        end;
    fdClose(FD);
    FD:=fdOpen('test.dat',open_rdonly);
    { Read data again }
    If FD>0 then
      begin
      For I:=1 to 5 do
        if fdRead (FD,Data[1],10)<>10 then
          begin
          WriteLn ('Error when Reading !');
```

```
            Halt(2);
            end;
        fdCLose(FD);
        { Truncating file at 60 bytes }
        { For truncating, file must be open or write }
        FD:=fdOpen('test.dat',open_wronly,octal(666));
        if FD>0 then
          begin
          if not fdTruncate(FD,60) then
             WriteIn('Error when truncating !');
          fdClose (FD);
          end;
        end;
    end;
end.
```

23.12.30 fdSeek

Synopsis: Set file pointer position.

Declaration: `function fdSeek(fd: LongInt;pos: LongInt;seektype: LongInt) : LongInt`

Visibility: default

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.

For an example, see fdOpen (1010).

Errors: Errors are returned in LinuxError.

See also: fdOpen (1010), fdWrite (1013), fdClose (1009), fdRead (1011), fdTruncate (1012), fdFlush (1009)

23.12.31 fdTruncate

Synopsis: Truncate file on certain size.

Declaration: `function fdTruncate(fd: LongInt;size: LongInt) : Boolean`

Visibility: default

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 (1010), fdClose (1009), fdRead (1011), fdWrite (1013), fdFlush (1009), fdSeek (1012)

23.12.32 fdWrite

Synopsis: Write data to file descriptor

Declaration: `function fdWrite(fd: LongInt;const buf;size: LongInt) : LongInt`

Visibility: default

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 (1010), fdClose (1009), fdRead (1011), fdTruncate (1012), fdSeek (1012), fdFlush (1009)

23.12.33 FD_Clr

Synopsis: Clears a filedescriptor in a set

Declaration: `procedure FD_Clr(fd: LongInt;var fds: fdSet)`

Visibility: default

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

For an example, see Select (1043).

Errors: None.

See also: Select (1043), SelectText (1045), GetFS (1023), FD_ZERO (1014), FD_Set (1013), FD_IsSet (1013)

23.12.34 FD_IsSet

Synopsis: Check whether a filedescriptor is set

Declaration: `function FD_IsSet(fd: LongInt;var fds: fdSet) : Boolean`

Visibility: default

Description: `FD_Set` Checks whether file descriptor `fd` in filedescriptor set `fds` is set.

For an example, see Select (1043).

Errors: None.

See also: Select (1043), SelectText (1045), GetFS (1023), FD_ZERO (1014), FD_Clr (1013), FD_Set (1013)

23.12.35 FD_Set

Synopsis: Set a filedescriptor in a set

Declaration: `procedure FD_Set(fd: LongInt;var fds: fdSet)`

Visibility: default

Description: `FD_Set` sets file descriptor `fd` in filedescriptor set `fds`.

For an example, see Select (1043).

Errors: None.

See also: Select (1043), SelectText (1045), GetFS (1023), FD_ZERO (1014), FD_Clr (1013), FD_IsSet (1013)

23.12.36 FD_Zero

Synopsis: Clear all file descriptors in set

Declaration: `procedure FD_Zero(var fds: fdSet)`

Visibility: default

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

For an example, see Select (1043).

Errors: None.

See also: Select (1043), SelectText (1045), GetFS (1023), FD_Clr (1013), FD_Set (1013), FD_IsSet (1013)

23.12.37 FExpand

Synopsis: Expand filename to fully qualified path

Declaration: `function FExpand(const Path: PathStr) : PathStr`

Visibility: default

Description: `FExpand` expands `Path` to a full path, starting from root, eliminating directory references such as . and .. from the result.

Errors: None

See also: BaseName (993), DirName (1000)

Listing: ./olinuxex/ex45.pp

```
Program Example45;

{ Program to demonstrate the FExpand function. }

Uses oldlinux;

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

23.12.38 Flock

Synopsis: Lock a file (advisory lock)

Declaration:
```
function Flock(fd: LongInt;mode: LongInt) : Boolean
function Flock(var T: text;mode: LongInt) : Boolean
function Flock(var F: File;mode: LongInt) : Boolean
```

Visibility: default

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_SHsets a shared lock.

LOCK_EXsets an exclusive lock.

LOCK_UNunlocks the file.

LOCK_NB This can be OR-ed together with the other. If this is done the application doesn't block when locking.

The function returns `True` if successful, `False` otherwise.

Errors: If an error occurs, it is reported in `LinuxError`.

See also: Fcntl (1008)

23.12.39 FNMatch

Synopsis: Check whether filename matches wildcard specification

Declaration: `function FNMatch(const Pattern: String;const Name: String) : Boolean`

Visibility: default

Description: `FNMatch` returns `True` if the filename in `Name` matches the wildcard pattern in `Pattern`, `False` otherwise.

`Pattern` can contain the wildcards * (match zero or more arbitrary characters) or `?` (match a single character).

Errors: None.

See also: FSearch (1017), FExpand (1014)

Listing: ./olinuxex/ex69.pp

```
Program Example69;

{ Program to demonstrate the FNMatch function. }

Uses oldlinux;

  Procedure TestMatch(Pattern,Name : String);

  begin
    Write ('"',Name,'" ');
    If FNMatch (Pattern,Name) then
      Write ('matches')
    else
      Write ('does not match');
    Writeln(' "',Pattern,'".');
  end;

begin
  TestMatch('*','FileName');
  TestMatch('.*','FileName');
  TestMatch('*a*','FileName');
  TestMatch('?ile*','FileName');
  TestMatch('?','FileName');
  TestMatch('.?','FileName');
  TestMatch('?a*','FileName');
  TestMatch('??*me?','FileName');
end.
```

23.12.40 Fork

Synopsis: Create child process

Declaration:
```
function Fork : LongInt
```

Visibility: default

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 (1026)).

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 (1006), Clone (997)

23.12.41 FReName

Synopsis: Rename file

Declaration:
```
function FReName(OldName: Pchar;NewName: Pchar) : Boolean
function FReName(OldName: String;NewName: String) : Boolean
```

Visibility: default

Description: `FRename` renames the file `OldName` to `NewName`. `NewName` can be in a different directory than `OldName`, but it cannot be on another partition (device). Any existing file on the new location will be replaced.

If the operation fails, then the `OldName` file will be preserved.

The function returns `True` on succes, `False` on failure.

Errors: On error, errors are reported in `LinuxError`. Possible errors include:

sys_eisdir`NewName` exists and is a directory, but `OldName` is not a directory.

sys_exdev`NewName` and `OldName` are on different devices.

sys_enotempty or sys_eexist`NewName` is an existing, non-empty directory.

sys_ebusy`OldName` or `NewName` is a directory and is in use by another process.

sys_einval`NewName` is part of `OldName`.

sys_emlink`OldPath` or `NewPath` already have tha maximum amount of links pointing to them.

sys_enotdirpart of `OldName` or `NewName` is not directory.

sys_efault For the `pchar` case: One of the pointers points to an invalid address.

sys_eaccessaccess is denied when attempting to move the file.

sys_enametoolong Either `OldName` or `NewName` is too long.

sys_enoenta directory component in `OldName` or `NewName` didn't exist.

sys_enomemnot enough kernel memory.

sys_erofs`NewName` or `OldName` is on a read-only file system.

sys_elooptoo many symbolic links were encountered trying to expand `OldName` or `NewName`

sys_enospcthe filesystem has no room for the new directory entry.

See also: UnLink (1061)

23.12.42 FSearch

Synopsis: Search for file in search path.

Declaration: `function FSearch(const path: PathStr;dirlist: String) : PathStr`

Visibility: default

Description: `FSearch` 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 (993), DirName (1000), FExpand (1014), FNMatch (1015)

Listing: ./olinuxex/ex46.pp

```
Program Example46;

{ Program to demonstrate the FSearch function. }

Uses oldlinux,strings;

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

23.12.43 FSplit

Synopsis: Split filename into path, name and extension

Declaration: `procedure FSplit(const Path: PathStr;var Dir: DirStr;var Name: NameStr;`
`var Ext: ExtStr)`

Visibility: default

Description: `FSplit` splits a full file name into 3 parts : A `Path`, a `Name` and an extension (in `ext`). The extension is taken to be all letters after the last dot (.).

Errors: None.

See also: FSearch (1017)

Listing: ./olinuxex/ex67.pp

```
Program Example67;

uses oldlinux;

{ Program to demonstrate the FSplit function. }

var
  Path,Name,Ext : string;

begin
  FSplit(ParamStr(1),Path,Name,Ext);
  WriteLn('Split ',ParamStr(1),' in:');
  WriteLn('Path      : ',Path);
  WriteLn('Name      : ',Name);
  WriteLn('Extension : ',Ext);
end.
```

23.12.44 FSStat

Synopsis: Retrieve filesystem information.

Declaration:
```
function FSStat(Path: PathStr;var Info: Statfs) : Boolean
function FSStat(Fd: LongInt;var Info: Statfs) : Boolean
```

Visibility: default

Description: `FSStat` returns in `Info` information about the filesystem on which the file `Path` resides, or on which the file with file descriptor `fd` resides. Info is of type `statfs`. The function returns `True` if the call was succesfull, `False` if the call failed.

Errors: `LinuxError` is used to report errors.

sys_enotdir A 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_eloop A circular symbolic link was encountered in `Path`.

sys_eio An error occurred while reading from the filesystem.

See also: FStat (1019), LStat (1033)

Listing: ./olinuxex/ex30.pp

```
program Example30;

{ Program to demonstrate the FSStat function. }

uses oldlinux;

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.
```

23.12.45 FStat

Synopsis: Retrieve information about a file

Declaration:
```
function FStat(Path: PathStr;var Info: Stat) : Boolean
function FStat(Fd: LongInt;var Info: Stat) : Boolean
function FStat(var F: Text;var Info: Stat) : Boolean
function FStat(var F: File;var Info: Stat) : Boolean
```

Visibility: default

Description: `FStat` gets information about the file specified in one of the following:

Path a file on the filesystem.

Fd a valid file descriptor.

F an opened text file or untyped file.

and stores it in `Info`, which is of type `stat`. The function returns `True` if the call was succesfull, `False` if the call failed.

Errors: `LinuxError` is used to report errors.

sys_enoent `Path` does not exist.

See also: FSStat (1018), LStat (1033)

Listing: ./olinuxex/ex28.pp

```
program example28;

{ Program to demonstrate the FStat function. }

uses oldlinux;

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.
```

23.12.46 GetDate

Synopsis: Return the system date

Declaration: `procedure GetDate(var Year: Word;var Month: Word;var Day: Word)`

Visibility: default

Description: Returns the current date.

Errors: None

See also: GetEpochTime (1022), GetTime (1027), GetDateTime (1020), EpochToLocal (1002)

Listing: ./olinuxex/ex6.pp

```
Program Example6;

{ Program to demonstrate the GetDate function. }

Uses oldlinux;

Var Year, Month, Day : Word;

begin
 GetDate (Year, Month, Day);
 WriteLn ('Date : ',Day:2,'/',Month:2,'/',Year:4);
end.
```

23.12.47 GetDateTime

Synopsis: Return system date and time

Declaration: `procedure GetDateTime(var Year: Word;var Month: Word;var Day: Word;`
`var hour: Word;var minute: Word;var second: Word)`

Visibility: default

Description: Returns the current date and time. The time is corrected for the local time zone. This procedure is equivalent to the GetDate (1020) and `GetTime` calls.

Errors: None

See also: GetEpochTime (1022), GetTime (1027), EpochToLocal (1002), GetDate (1020)

Listing: ./olinuxex/ex60.pp

```
Program Example6;

{ Program to demonstrate the GetDateTime function. }

Uses oldlinux;

Var Year, Month, Day, Hour, min, sec : Word;

begin
 GetDateTime (Year, Month, Day, Hour, min, sec);
 Writeln ('Date : ',Day:2,'/',Month:2,'/',Year:4);
 Writeln ('Time : ',Hour:2,':',Min:2,':',Sec:2);
end.
```

23.12.48 GetDomainName

Synopsis: Return current domain name

Declaration: `function GetDomainName : String`

Visibility: default

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 (1024)

Listing: ./olinuxex/ex39.pp

```
Program Example39;

{ Program to demonstrate the GetDomainName function. }

Uses oldlinux;

begin
  Writeln ('Domain name of this machine is : ',GetDomainName);
end.
```

23.12.49 GetEGid

Synopsis: Return effective group ID

Declaration: `function GetEGid : LongInt`

Visibility: default

Description: Get the effective group ID of the currently running process.

Errors: None.

See also: GetGid (1024)

Listing: ./olinuxex/ex18.pp

```
Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses oldlinux;

begin
 writeln ('Group Id = ',getgid,' Effective group Id = ',getegid);
end.
```

23.12.50 GetEnv

Synopsis: Return value of environment variable.

Declaration: `function GetEnv(P: String) : Pchar`

Visibility: default

Description: `GetEnv` 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.

Listing: ./olinuxex/ex41.pp

```
Program Example41;

{ Program to demonstrate the GetEnv function. }

Uses oldlinux;

begin
  WriteIn ('Path is : ',Getenv('PATH'));
end.
```

23.12.51 GetEpochTime

Synopsis: Return the current unix time

Declaration: `function GetEpochTime : LongInt`

Visibility: default

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 (1002), GetTime (1027)

Listing: ./olinuxex/ex1.pp

```
Program Example1;

{ Program to demonstrate the GetEpochTime function. }

Uses oldlinux;

begin
  Write ('Secs past the start of the Epoch (00:00 1/1/1980) : ');
  WriteIn (GetEpochTime);
end.
```

23.12.52 GetEUid

Synopsis: Return effective user ID

Declaration: `function GetEUid : LongInt`

Visibility: default

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

Errors: None.

See also: GetUid (1028)

Listing: ./olinuxex/ex17.pp

```
Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses oldlinux;

begin
  writeln ('User Id = ',getuid,' Effective user Id = ',geteuid);
end.
```

23.12.53 GetFS

Synopsis: Return file selector

Declaration:
```
function GetFS(var T: Text) : LongInt
function GetFS(var F: File) : LongInt
```

Visibility: default

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 (1043) call or so.

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

See also: Select (1043)

Listing: ./olinuxex/ex34.pp

```
Program Example33;

{ Program to demonstrate the SelectText function. }

Uses oldlinux;

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.
```

23.12.54 GetGid

Synopsis: Return real group ID

Declaration: `function GetGid : LongInt`

Visibility: default

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

Errors: None.

See also: GetEGid (1021)

Listing: ./olinuxex/ex18.pp

```
Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses oldlinux;

begin
 writeln ('Group Id = ',getgid,' Effective group Id = ',getegid);
end.
```

23.12.55 GetHostName

Synopsis: Return host name

Declaration: `function GetHostName : String`

Visibility: default

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 (1021)

Listing: ./olinuxex/ex40.pp

```
Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses oldlinux;

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

23.12.56 GetLocalTimezone

Synopsis: Return local timzeone information

Declaration:
```
procedure GetLocalTimezone(timer: LongInt;var leap_correct: LongInt;
                           var leap_hit: LongInt)
procedure GetLocalTimezone(timer: LongInt)
```

Visibility: default

Description: `GetLocalTimeZone` returns the local timezone information. It also initializes the `TZSeconds` variable, which is used to correct the epoch time to local time.

There should never be any need to call this function directly. It is called by the initialization routines of the Linux unit.

See also: GetTimezoneFile (1028), ReadTimezoneFile (1043)

23.12.57 GetPid

Synopsis: Return current process ID

Declaration:
```
function GetPid : LongInt
```

Visibility: default

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

Errors: None.

See also: GetPPid (1026)

Listing: ./olinuxex/ex16.pp

```
Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses oldlinux;
```

```
begin
  WriteIn ('Process Id = ',getpid,' Parent process Id = ',getppid);
end.
```

23.12.58 GetPPid

Synopsis: Return parent process ID

Declaration: `function GetPPid : LongInt`

Visibility: default

Description: Get the Process ID of the parent process.

Errors: None.

See also: GetPid (1025)

Listing: ./olinuxex/ex16.pp

```
Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses oldlinux;

begin
  WriteIn ('Process Id = ',getpid,' Parent process Id = ',getppid);
end.
```

23.12.59 GetPriority

Synopsis: Return process priority

Declaration: `function GetPriority(Which: Integer;Who: Integer) : Integer`

Visibility: default

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.

For an example, see Nice (1038).

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 (1046), Nice (1038)

23.12.60 GetTime

Synopsis: Return current system time

Declaration:
```
procedure GetTime(var hour: Word;var min: Word;var sec: Word;
                  var msec: Word;var usec: Word)
procedure GetTime(var hour: Word;var min: Word;var sec: Word;
                  var sec100: Word)
procedure GetTime(var hour: Word;var min: Word;var sec: Word)
```

Visibility: default

Description: Returns the current time of the day, adjusted to local time. Upon return, the parameters are filled with

hourHours since 00:00 today.

minminutes in current hour.

secseconds in current minute.

sec100hundreds of seconds in current second.

msecmilliseconds in current second.

usecmicroseconds in current second.

Errors: None

See also: GetEpochTime (1022), GetDate (1020), GetDateTime (1020), EpochToLocal (1002)

Listing: ./olinuxex/ex5.pp

```
Program Example5;

{ Program to demonstrate the GetTime function. }

Uses oldlinux;

Var Hour, Minute, Second : Word;

begin
  GetTime (Hour, Minute, Second);
  WriteIn ('Time : ',Hour:2,':',Minute:2,':',Second:2);
end.
```

23.12.61 GetTimeOfDay

Synopsis: Return kernel time of day in GMT

Declaration:
```
procedure GetTimeOfDay(var tv: timeval)
function GetTimeOfDay : LongInt
```

Visibility: default

Description: `GetTimeOfDay` returns the number of seconds since 00:00, January 1 1970, GMT in a `timeval` record. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call. To get the local time, GetTime (1027).

Errors: None.

See also: GetTime (1027), GetTimeOfDay (1027)

23.12.62 GetTimezoneFile

Synopsis: Return name of timezone information file

Declaration: `function GetTimezoneFile : String`

Visibility: default

Description: `GetTimezoneFile` returns the location of the current timezone file. The location of file is determined as follows:

1. If /etc/timezone exists, it is read, and the contents of this file is returned. This should work on Debian systems.
2. If /usr/lib/zoneinfo/localtime exists, then it is returned. (this file is a symlink to the timezone file on SuSE systems)
3. If /etc/localtime exists, then it is returned. (this file is a symlink to the timezone file on RedHat systems)

Errors: If no file was found, an empty string is returned.

See also: ReadTimezoneFile (1043)

23.12.63 GetUid

Synopsis: Return current user ID

Declaration: `function GetUid : LongInt`

Visibility: default

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

Errors: None.

See also: GetEUid (1023)

Listing: ./olinuxex/ex17.pp

```
Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses oldlinux;

begin
  writeln ('User Id = ',getuid,' Effective user Id = ',geteuid);
end.
```

23.12.64 Glob

Synopsis: Find filenames matching a wildcard pattern

Declaration: `function Glob(const path: PathStr) : pglob`

Visibility: default

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 (1029)

Listing: ./olinuxex/ex49.pp

```
Program Example49;

{ Program to demonstrate the Glob and GlobFree functions. }

Uses oldlinux;

Var G1,G2 : PGlob;

begin
  G1:=Glob ('*');
  if LinuxError=0 then
    begin
    G2:=G1;
    WriteIn ('Files in this directory : ');
    While g2<>Nil do
      begin
      WriteIn (g2^.name);
      g2:=g2^.next;
      end;
    GlobFree (g1);
    end;
end.
```

23.12.65 Globfree

Synopsis: Free result of Glob (1028) call

Declaration: `procedure Globfree(var p: pglob)`

Visibility: default

Description: Releases the memory, occupied by a pglob structure. `P` is set to nil.

For an example, see Glob (1028).

Errors: None

See also: Glob (1028)

23.12.66 IOCtl

Synopsis: General kernel IOCTL call.

Declaration: `function IOCtl(Handle: LongInt;Ndx: LongInt;Data: Pointer) : Boolean`

Visibility: default

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

Listing: ./olinuxex/ex54.pp

```
Program Example54;

uses oldlinux;

{ 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.
```

23.12.67 IOperm

Synopsis: Set permission on IO ports

Declaration: `function IOperm(From: Cardinal;Num: Cardinal;Value: LongInt) : Boolean`

Visibility: default

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

Note:

- This works ONLY as root.
- Only the first `0x03ff` ports can be set.
- When doing a Fork (1016), the permissions are reset. When doing a Execve (1006) they are kept.

Errors: Errors are returned in `LinuxError`

23.12.68 IoPL

Synopsis: Set I/O privilege level

Declaration: `function IoPL(Level: LongInt) : Boolean`

Visibility: default

Description: `IoPL` sets the I/O privilige level. It is intended for completeness only, one should normally not use it.

23.12.69 IsATTY

Synopsis: Check if filehandle is a TTY (terminal)

Declaration:
```
function IsATTY(Handle: LongInt) : Boolean
function IsATTY(var f: text) : Boolean
```

Visibility: default

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 (1029), TTYName (1060)

23.12.70 Kill

Synopsis: Send a signal to a process

Declaration:
```
function Kill(Pid: LongInt;Sig: LongInt) : Integer
```

Visibility: default

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 (1047), Signal (1048)

23.12.71 Link

Synopsis: Create a hard link to a file

Declaration:
```
function Link(OldPath: PathStr;NewPath: PathStr) : Boolean
```

Visibility: default

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 another entry.

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

See also: SymLink (1052), UnLink (1061)

Listing: ./olinuxex/ex21.pp

```
Program Example21;

{ Program to demonstrate the Link and UnLink functions. }

Uses oldlinux;

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.
```

23.12.72 LocalToEpoch

Synopsis: Convert local time to epoch (unix) time

Declaration: `function LocalToEpoch(year: Word;month: Word;day: Word;hour: Word; minute: Word;second: Word) : LongInt`

Visibility: default

Description: Converts the Local time to epoch time (=Number of seconds since 00:00:00 , January 1, 1970).

Errors: None

See also: GetEpochTime (1022), EpochToLocal (1002), GetTime (1027), GetDate (1020)

Listing: ./olinuxex/ex4.pp

```
Program Example4;

{ Program to demonstrate the LocalToEpoch function. }

Uses oldlinux;

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.
```

23.12.73 Lstat

Synopsis: Return information about symbolic link. Do not follow the link

Declaration: `function Lstat(Filename: PathStr;var Info: Stat) : Boolean`

Visibility: default

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 (1019), FSStat (1018)

Listing: ./olinuxex/ex29.pp

```
program example29;

{ Program to demonstrate the LStat function. }

uses oldlinux;
```

```
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.
```

23.12.74 mkFifo

Synopsis: Create FIFO (named pipe) in file system

Declaration: `function mkFifo(pathname: String;mode: LongInt) : Boolean`

Visibility: default

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 (1040), MkFifo (1035)

23.12.75 MMap

Synopsis: Create memory map of a file

Declaration: `function MMap(const m: tmmapargs) : LongInt`

Visibility: default

Description: `MMap` maps or unmaps files or devices into memory. The different fields of the argument `m` determine what and how the `mmap` maps this:

address Address where to mmap the device. This address is a hint, and may not be followed.

size Size (in bytes) of area to be mapped.

prot Protection of mapped memory. This is a OR-ed combination of the following constants:

- **PROT_EXEC** The memory can be executed.
- **PROT_READ** The memory can be read.
- **PROT_WRITE** The memory can be written.
- **PROT_NONE** The memory can not be accessed.

flags Contains some options for the mmap call. It is an OR-ed combination of the following constants:

- **MAP_FIXED** Do not map at another address than the given address. If the address cannot be used, `MMap` will fail.
- **MAP_SHARED** Share this map with other processes that map this object.
- **MAP_PRIVATE** Create a private map with copy-on-write semantics.
- **MAP_ANONYMOUS** `fd` does not have to be a file descriptor.

One of the options `MAP_SHARED` and `MAP_PRIVATE` must be present, but not both at the same time.

fd File descriptor from which to map.

offset Offset to be used in file descriptor fd.

The function returns a pointer to the mapped memory, or a -1 in case of en error.

Errors: On error, -1 is returned and LinuxError is set to the error code:

Sys_EBADF `fd` is not a valid file descriptor and `MAP_ANONYMOUS` was not specified.

Sys_EACCES `MAP_PRIVATE` was specified, but fd is not open for reading. Or `MAP_SHARED` was asked and `PROT_WRITE` is set, fd is not open for writing

Sys_EINVAL One of the record fields `Start`, `length` or `offset` is invalid.

Sys_ETXTBUSY`MAP_DENYWRITE` was set but the object specified by fd is open for writing.

Sys_EAGAIN`fd` is locked, or too much memory is locked.

Sys_ENOMEM Not enough memory for this operation.

See also: MUnMap (1036)

Listing: ./olinuxex/ex66.pp

```
Program Example66;

{ Program to demonstrate the MMap function. }

Uses oldlinux;

Var S : String;
    fd,Len : Longint;
    args : tmmapargs;
    P : PChar;

begin
  S:='This is a string'#0;
  Len:=Length(S);
  fd:=fdOpen('testfile.txt',Open_wrOnly or open_creat);
  If fd=-1 then
    Halt(1);
  If fdWrite(fd,S[1],Len)=-1 then
    Halt(2);
  fdClose(fd);
  fdOpen('testfile.txt',Open_rdOnly);
  if fd=-1 then
    Halt(3);
  args.address:=0;
  args.offset:=0;
  args.size:=Len+1;
  args.fd:=Fd;
  args.flags:=MAP_PRIVATE;
  args.prot:=PROT_READ or PROT_WRITE;
  P:=Pchar(mmap(args));
  If longint(P)=-1 then
    Halt(4);
  Writeln('Read in memory  :',P);
  fdclose(fd);
  if Not MUnMap(P,Len) Then
    Halt(LinuxError);
end.
```

23.12.76 MUnMap

Synopsis: Unmap previously mapped memory block

Declaration: `function MUnMap(P: Pointer;Size: LongInt) : Boolean`

Visibility: default

Description: `MUnMap` unmaps the memory block of size `Size`, pointed to by `P`, which was previously allocated with MMap (1035).

The function returns `True` if successful, `False` otherwise.

For an example, see MMap (1035).

Errors: In case of error the function returns `False` and `LinuxError` is set to an error value. See MMap (1035) for possible error values.

See also: MMap (1035)

23.12.77 NanoSleep

Synopsis: Suspend process for a short time

Declaration: `function NanoSleep(const req: timespec;var rem: timespec) : LongInt`

Visibility: default

Description: `NanoSleep` suspends the process till a time period as specified in `req` has passed. Then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and `rem` will contain the remaining time till the end of the intended period. In this case the return value will be -1, and `LinuxError` will be set to `EINTR`

If the function returns without error, the return value is zero.

Errors: If the call was interrupted, -1 is returned, and `LinuxError` is set to `EINTR`. If invalid time values were specified, then -1 is returned and `LinuxError` is set to `EINVAL`.

See also: Pause (1040), Alarm (990)

Listing: ./olinuxex/ex72.pp

```
program example72;

{ Program to demonstrate the NanoSleep function. }

uses oldlinux;

Var
  Req,Rem : TimeSpec;
  Res : Longint;

begin
  With Req do
    begin
    tv_sec:=10;
    tv_nsec:=100;
    end;
  Write('NanoSleep returned : ');
  Flush(Output);
  Res:=(NanoSleep(Req,rem));
  Writeln(res);
  If (res<>0) then
    With rem do
      begin
      Writeln('Remaining seconds      : ',tv_sec);
      Writeln('Remaining nanoseconds : ',tv_nsec);
      end;
end.
```

23.12.78 Nice

Synopsis: Set process priority

Declaration: `procedure Nice(N: Integer)`

Visibility: default

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 (1026), SetPriority (1046)

Listing: ./olinuxex/ex15.pp

```
Program Example15;

{ Program to demonstrate the Nice and Get/SetPriority functions. }

Uses oldlinux;

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.
```

23.12.79 Octal

Synopsis: Convert octal to decimal value

Declaration: `function Octal(l: LongInt) : LongInt`

Visibility: default

Description: `Octal` will convert a number specified as an octal number to it's decimal value.

This is useful for the Chmod (995) call, where permissions are specified as octal numbers.

Errors: No checking is performed whether the given number is a correct Octal number. e.g. specifying `998` is possible; the result will be wrong in that case.

See also: Chmod (995)

Listing: ./olinuxex/ex68.pp

```
Program Example68;

{ Program to demonstrate the Octal function. }

Uses oldlinux;
```

```
begin
  Writeln('Mode 777 : ', Octal(777));
  Writeln('Mode 644 : ', Octal(644));
  Writeln('Mode 755 : ', Octal(755));
end.
```

23.12.80 OpenDir

Synopsis: Open directory for reading

Declaration:
```
function OpenDir(f: pchar) : PDir
function OpenDir(f: String) : PDir
```

Visibility: default

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 (999), ReadDir (1041), SeekDir (1043), TellDir (1060)

Listing: ./olinuxex/ex35.pp

```
Program Example35;

{ Program to demonstrate the
  OpenDir,ReadDir, SeekDir and TellDir functions. }

Uses oldlinux;

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.
```

23.12.81 Pause

Synopsis: Wait for a signal

Declaration: `procedure Pause`

Visibility: default

Description: `Pause` puts the process to sleep and waits until the application receives a signal. If a signal handler is installed for the received sigal, the handler will be called and after that pause will return control to the process.

For an example, see Alarm (990).

23.12.82 PClose

Synopsis: Close file opened with POpen (1040)

Declaration:
```
function PClose(var F: text) : LongInt
function PClose(var F: File) : LongInt
```

Visibility: default

Description: `PClose` closes a file opened with POpen (1040). It waits for the command to complete, and then returns the exit status of the command.

For an example, see POpen (1040)

Errors: `LinuxError` is used to report errors. If it is different from zero, the exit status is not valid.

See also: POpen (1040)

23.12.83 POpen

Synopsis: Pipe file to standard input/output of program

Declaration:
```
procedure POpen(var F: text;const Prog: String;rw: Char)
procedure POpen(var F: File;const Prog: String;rw: Char)
```

Visibility: default

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 (1040). 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 (991), PClose (1040)

Listing: ./olinuxex/ex37.pp

```
Program Example37;

{ Program to demonstrate the Popen function. }

uses oldlinux;

var f : text;
    i : longint;

begin
  writeln ('Creating a shell script to which echoes its arguments');
  writeln ('and input back to stdout');
  assign (f,'test21a');
  rewrite (f);
  writeln (f,'#!/bin/sh');
  writeln (f,'echo this is the child speaking.... ');
  writeln (f,'echo got arguments \*"$*"\*');
  writeln (f,'cat');
  writeln (f,'exit 2');
  writeln (f);
  close (f);
  chmod ('test21a',octal (755));
  popen (f,'./test21a arg1 arg2','W');
  if linuxerror<>0 then
     writeln ('error from POpen : Linuxerror : ', Linuxerror);
  for i:=1 to 10 do
    writeln (f,'This is written to the pipe, and should appear on stdout.');
  Flush(f);
  Writeln ('The script exited with status : ',PClose (f));
  writeln;
  writeln ('Press <return> to remove shell script.');
  readln;
  assign (f,'test21a');
  erase (f)
end.
```

23.12.84 ReadDir

Synopsis: Read entry from directory

Declaration: `function ReadDir(p: PDir) : pdirent`

Visibility: default

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.

For an example, see OpenDir (1039).

Errors: Errors are returned in LinuxError.

See also: CloseDir (999), OpenDir (1039), SeekDir (1043), TellDir (1060)

23.12.85 ReadLink

Synopsis: Read destination of symbolic link

Declaration:
```
function ReadLink(name: pchar;linkname: pchar;maxlen: LongInt) : LongInt
function ReadLink(name: PathStr) : PathStr
```

Visibility: default

Description: `ReadLink` returns the file the symbolic link `name` is pointing to. The first form of this function accepts a buffer `linkname` of length `maxlen` where the filename will be stored. It returns the actual number of characters stored in the buffer.

The second form of the function returns simply the name of the file.

Errors: On error, the first form of the function returns -1; the second one returns an empty string. `LinuxError` is set to report errors:

SYS_ENOTDIR A part of the path in `Name` is not a directory.

SYS_EINVAL maxlen is not positive, or the file is not a symbolic link.

SYS_ENAMETOOLONG A pathname, or a component of a pathname, was too long.

SYS_ENOENT the link `name` does not exist.

SYS_EACCES No permission to search a directory in the path

SYS_ELOOP Too many symbolic links were encountered in translating the pathname.

SYS_EIO An I/O error occurred while reading from the file system.

SYS_EFAULT The buffer is not part of the the process's memory space.

SYS_ENOMEM Not enough kernel memory was available.

See also: SymLink (1052)

Listing: ./olinuxex/ex62.pp

```
Program Example62;

{ Program to demonstrate the ReadLink function. }

Uses oldlinux;

Var F : Text;
    S : String;

begin
  Assign (F,'test.txt');
  Rewrite (F);
  WriteIn (F,'This is written to test.txt');
```

```
  Close(f);
  { new.txt and test.txt are now the same file }
  if not SymLink ('test.txt','new.txt') then
    writeln ('Error when symlinking !');
  S:=ReadLink('new.txt');
  If S='' then
    WriteIn ('Error reading link !')
  Else
    WriteIn ('Link points to : ',S);
 { Now remove links }
 If not Unlink ('new.txt') then
   WriteIn ('Error when unlinking !');
 If not Unlink ('test.txt') then
   WriteIn ('Error when unlinking !');
end.
```

23.12.86 ReadTimezoneFile

Synopsis: Read the timezone file and initialize time routines

Declaration: `procedure ReadTimezoneFile(fn: String)`

Visibility: default

Description: `ReadTimeZoneFile` reads the timezone file `fn` and initializes the local time routines based on the information found there.

There should be no need to call this function. The initialization routines of the linux unit call this routine at unit startup.

Errors: None.

See also: GetTimezoneFile (1028), GetLocalTimezone (1025)

23.12.87 SeekDir

Synopsis: Seek to position in directory

Declaration: `procedure SeekDir(p: PDir;off: LongInt)`

Visibility: default

Description: `SeekDir` sets the directory pointer to the `off`-th entry in the directory structure pointed to by `p`.

For an example, see OpenDir (1039).

Errors: Errors are returned in LinuxError.

See also: CloseDir (999), ReadDir (1041), OpenDir (1039), TellDir (1060)

23.12.88 Select

Synopsis: Wait for events on file descriptors

Declaration:
```
function Select(N: LongInt;readfds: pfdset;writefds: pfdset;
                exceptfds: pfdset;TimeOut: ptimeval) : LongInt
function Select(N: LongInt;readfds: pfdset;writefds: pfdset;
                exceptfds: pfdset;TimeOut: LongInt) : LongInt
```

Visibility: default

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_ZERO (1014)FD_Clr (1013), FD_Set (1013) or FD_IsSet (1013) 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 descriptor was specified in one of the sets.

SYS_EINTR A non blocked signal was caught.

SYS_EINVAL `N` is negative or too big.

SYS_ENOMEM `Select` was unable to allocate memory for its internal tables.

See also: SelectText (1045), GetFS (1023), FD_ZERO (1014), FD_Clr (1013), FD_Set (1013), FD_IsSet (1013)

Listing: ./olinuxex/ex33.pp

```
Program Example33;

{ Program to demonstrate the Select function. }

Uses oldlinux;

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
    WriteIn ('Thank you !')
    { FD_ISSET(0,FDS) would be true here. }
  else
    WriteIn ('Too late !');
end.
```

23.12.89 SelectText

Synopsis: Wait for event on typed ontyped file.

Declaration:
```
function SelectText(var T: Text;TimeOut: ptimeval) : LongInt
function SelectText(var T: Text;TimeOut: LongInt) : LongInt
```

Visibility: default

Description: `SelectText` executes the Select (1043) 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 (1043). `SYS_EBADF` can also mean that the file wasn't opened.

See also: Select (1043), GetFS (1023)

23.12.90 SetDate

Synopsis: Set the current system date.

Declaration:
```
function SetDate(Year: Word;Month: Word;Day: Word) : Boolean
```

Visibility: default

Description: `SetDate` sets the system date to `year`, `month`, `day`. This is the kernel date, so it is in GMT. The time is not touched. The function returns `True` if the call was executed corretly, `False` otherwise.

Remark: You must be root to execute this call.

Errors: Errors are returned in LinuxError (989)

See also: GetDate (1020), SetTime (1046), SetDateTime (1045)

23.12.91 SetDateTime

Synopsis: Set the current system date and time

Declaration:
```
function SetDateTime(Year: Word;Month: Word;Day: Word;hour: Word;
                     minute: Word;second: Word) : Boolean
```

Visibility: default

Description: `SetDate` sets the system date and time to `year`, `month`, `day`, `hour`, `min`, `Sec`. This is the kernel date/time, so it is in GMT. The time is not touched. The function returns `True` if the call was executed corretly, `False` otherwise.

Remark: You must be root to execute this call.

Errors: Errors are returned in LinuxError (989)

See also: SetDate (1045), SetTime (1046), GetDateTime (1020)

23.12.92 SetPriority

Synopsis: Set process priority

Declaration:
```
procedure SetPriority(Which: Integer;Who: Integer;What: Integer)
```

Visibility: default

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 constants:

Prio_Process`Who` is interpreted as process ID

Prio_PGrp`Who` is interpreted as process group ID

Prio_User`Who` is interpreted as user ID

`Prio` is a value in the range -20 to 20.

For an example, see Nice (1038).

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 (1026), Nice (1038)

23.12.93 SetTime

Synopsis: Set the current system time.

Declaration:
```
function SetTime(Hour: Word;Min: Word;Sec: Word) : Boolean
```

Visibility: default

Description: `SetTime` sets the system time to `hour`, `min`, `Sec`. This is the kernel time, so it is in GMT. The date is not touched. The function returns `True` if the call was executed corretly, `False` otherwise.

Remark: You must be root to execute this call.

Errors: Errors are returned in LinuxError (989)

See also: GetTime (1027), SetDate (1045), SetDateTime (1045)

23.12.94 Shell

Synopsis: Execute and feed command to system shell

Declaration:
```
function Shell(const Command: String) : LongInt
function Shell(const Command: AnsiString) : LongInt
```

Visibility: default

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 (1016) or Execve (1006) calls.

Errors: Errors are reported in LinuxError.

See also: POpen (1040), Fork (1016), Execve (1006)

Listing: ./olinuxex/ex56.pp

```
program example56;

uses oldlinux;

{ Program to demonstrate the Shell function }

Var S : Longint;

begin
  WriteIn ('Output of ls -l *.pp');
  S:=Shell ('ls -l *.pp');
  WriteIn ('Command exited wwith status : ',S);
end.
```

23.12.95 SigAction

Synopsis: Install signal handler

Declaration:
```
procedure SigAction(Signum: LongInt;Act: PSigActionRec;
                    OldAct: PSigActionRec)
```

Visibility: default

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 (1049), SigPending (1049), SigSuspend (1051), Kill (1031)

Listing: ./olinuxex/ex57.pp

```
Program example57;

{ Program to demonstrate the SigAction function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

uses oldlinux;

Var
   oa,na : PSigActionRec;

Procedure DoSig(sig : Longint);cdecl;

begin
   writeln('Receiving signal: ',sig);
end;

begin
   new(na);
   new(oa);
   na^.Handler.sh:=@DoSig;
   na^.Sa_Mask:=0;
   na^.Sa_Flags:=0;
   na^.Sa_Restorer:=Nil;
   SigAction(SigUsr1,na,oa);
   if LinuxError<>0 then
     begin
     writeln('Error: ',linuxerror,'.');
     halt(1);
     end;
   Writeln ('Send USR1 signal or press <ENTER> to exit');
   readln;
end.
```

23.12.96 Signal

Synopsis: Install signal handler (deprecated)

Declaration: `function Signal(Signum: LongInt;Handler: SignalHandler) : SignalHandler`

Visibility: default

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 (1047), Kill (1031)

Listing: ./olinuxex/ex58.pp

```
Program example58;

{ Program to demonstrate the Signal function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

uses oldlinux;

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.
```

23.12.97 SigPending

Synopsis: Return set of currently pending signals

Declaration: `function SigPending : SigSet`

Visibility: default

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 (1047), SigProcMask (1049), SigSuspend (1051), Signal (1048), Kill (1031)

23.12.98 SigProcMask

Synopsis: Set list of blocked signals

Declaration: `procedure SigProcMask(How: LongInt;SSet: PSigSet;OldSSet: PSigSet)`

Visibility: default

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 (1047), SigPending (1049), SigSuspend (1051), Kill (1031)

23.12.99 SigRaise

Synopsis: Raise a signal (send to current process)

Declaration: `procedure SigRaise(Sig: Integer)`

Visibility: default

Description: `SigRaise` sends a `Sig` signal to the current process.

Errors: None.

See also: Kill (1031), GetPid (1025)

Listing: ./olinuxex/ex65.pp

```
Program example64;

{ Program to demonstrate the SigRaise function.}

uses oldlinux;

Var
  oa,na : PSigActionRec;

Procedure DoSig(sig : Longint);cdecl;

begin
  writeln('Receiving signal: ',sig);
end;

begin
  new(na);
  new(oa);
  na^.handler.sh:=@DoSig;
  na^.Sa_Mask:=0;
  na^.Sa_Flags:=0;
  na^.Sa_Restorer:=Nil;
  SigAction(SigUsr1,na,oa);
  if LinuxError<>0 then
    begin
    writeln('Error: ',linuxerror,'.');
    halt(1);
    end;
  Writeln('Sending USR1 (',sigusr1,') signal to self.');
  SigRaise(sigusr1);
end.
```

23.12.100 SigSuspend

Synopsis: Set signal mask and suspend process till signal is received

Declaration:
```
procedure SigSuspend(Mask: SigSet)
```

Visibility: default

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 (1047), SigProcMask (1049), SigPending (1049), Signal (1048), Kill (1031)

23.12.101 StringToPPChar

Synopsis: Split string in list of null-terminated strings

Declaration:
```
function StringToPPChar(var S: String) : ppchar
function StringToPPChar(var S: AnsiString) : ppchar
function StringToPPChar(S: Pchar) : ppchar
```

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of pchars that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various `Exec` calls.

Errors: None.

See also: CreateShellArgV (999), Execve (1006), Execv (1005)

Listing: ./olinuxex/ex70.pp

```
Program Example70;

{ Program to demonstrate the StringToPPchar function. }

Uses oldlinux;

Var S : String;
    P : PPChar;
    I : longint;

begin
  // remark whitespace at end.
  S:='This is a string with words. ';
  P:=StringToPPChar(S);
  I:=0;
  While P[i]<>Nil do
    begin
    Writeln('Word ',i,' : ',P[i]);
    Inc(I);
```

```
    end;
  FreeMem(P,i*SizeOf(Pchar));
end.
```

23.12.102 SymLink

Synopsis: Create a symbolic link

Declaration: `function SymLink(OldPath: PathStr;NewPath: PathStr) : Boolean`

Visibility: default

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 does not support linking files.

sys_eaccess Write access for the directory containing `Newpath` is disallowed, or one of the directories in `OldPath` or `NewPath` has no search (=execute) permission.

sys_enoent A directory entry in `OldPath` or `NewPath` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in `OldPath` or `NewPath` is 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 another entry.

See also: Link (1031), UnLink (1061), ReadLink (1042)

Listing: ./olinuxex/ex22.pp

```
Program Example22;

{ Program to demonstrate the SymLink and UnLink functions. }

Uses oldlinux;

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
    WriteIn ('Error when unlinking !');
  Assign (f,'new.txt');
  { This should fail, since the symbolic link
    points to a non-existent file! }
  {$i-}
  Reset (F);
  {$i+}
  If IOResult=0 then
    WriteIn ('This shouldn''t happen');
 { Now remove new.txt also }
 If not Unlink ('new.txt') then
   WriteIn ('Error when unlinking !');
end.
```

23.12.103 SysCall

Synopsis: Execute system call.

Declaration: `function SysCall(callnr: LongInt;var regs: SysCallRegs) : LongInt`

Visibility: default

Description: `SysCall` can be used to execute a direct system call. The call parameters must be encoded in `regs` and the call number must be specified by `callnr`. The call result is returned, and any modified registers are in `regs`

Errors: None.

See also: SysCallregs (985)

23.12.104 Sysinfo

Synopsis: Return kernel system information

Declaration: `function Sysinfo(var Info: TSysinfo) : Boolean`

Visibility: default

Description: `SysInfo` returns system information in `Info`. Returned information in `Info` includes:

uptimeNumber of seconds since boot.

loads1, 5 and 15 minute load averages.

totalramtotal amount of main memory.

freeramamount of free memory.

sharedramamount of shared memory.

bufferramamount of memory used by buffers.

totalswaptotal amount of swapspace.

freeswapamount of free swapspace.

procsnumber of current processes.

Errors: None.

See also: Uname (1061)

Listing: ./olinuxex/ex64.pp

```
program Example64;

{ Example to demonstrate the SysInfo function }

Uses oldlinux;

Function Mb(L : Longint) : longint;

begin
  Mb:=L div (1024*1024);
end;

Var Info : TSysInfo;
    D,M,Secs,H : longint;

begin
  If Not SysInfo(Info) then
    Halt(1);
  With Info do
    begin
    D:=Uptime div (3600*24);
    UpTime:=UpTime mod (3600*24);
    h:=uptime div 3600;
    uptime:=uptime mod 3600;
    m:=uptime div 60;
    secs:=uptime mod 60;
    Writeln('Uptime : ',d,'days, ',h,' hours, ',m,' min, ',secs,' s.');
    Writeln('Loads  : ',Loads[1],'/',Loads[2],'/',Loads[3]);
    Writeln('Total Ram  : ',Mb(totalram),'Mb.');
    Writeln('Free Ram   : ',Mb(freeram),'Mb.');
    Writeln('Shared Ram : ',Mb(sharedram),'Mb.');
    Writeln('Buffer Ram : ',Mb(bufferram),'Mb.');
    Writeln('Total Swap : ',Mb(totalswap),'Mb.');
    Writeln('Free Swap  : ',Mb(freeswap),'Mb.');
    end;
end.
```

23.12.105 S_ISBLK

Synopsis: Is file a block device

Declaration: `function S_ISBLK(m: Word) : Boolean`

Visibility: default

Description: `S_ISBLK` checks the file mode `m` to see whether the file is a block device file. If so it returns `True`.

See also: FStat (1019), S_ISLNK (1055), S_ISREG (1056), S_ISDIR (1055), S_ISCHR (1055), S_ISFIFO (1055), S_ISSOCK (1056)

23.12.106 S_ISCHR

Synopsis: Is file a character device

Declaration: `function S_ISCHR(m: Word) : Boolean`

Visibility: default

Description: `S_ISCHR` checks the file mode `m` to see whether the file is a character device file. If so it returns `True`.

See also: FStat (1019), S_ISLNK (1055), S_ISREG (1056), S_ISDIR (1055), S_ISBLK (1054), S_ISFIFO (1055), S_ISSOCK (1056)

23.12.107 S_ISDIR

Synopsis: Is file a directory

Declaration: `function S_ISDIR(m: Word) : Boolean`

Visibility: default

Description: `S_ISDIR` checks the file mode `m` to see whether the file is a directory. If so it returns `True`

See also: FStat (1019), S_ISLNK (1055), S_ISREG (1056), S_ISCHR (1055), S_ISBLK (1054), S_ISFIFO (1055), S_ISSOCK (1056)

23.12.108 S_ISFIFO

Synopsis: Is file a FIFO

Declaration: `function S_ISFIFO(m: Word) : Boolean`

Visibility: default

Description: `S_ISFIFO` checks the file mode `m` to see whether the file is a fifo (a named pipe). If so it returns `True`.

See also: FStat (1019), S_ISLNK (1055), S_ISREG (1056), S_ISCHR (1055), S_ISBLK (1054), S_ISDIR (1055), S_ISSOCK (1056)

23.12.109 S_ISLNK

Synopsis: Is file a symbolic link

Declaration: `function S_ISLNK(m: Word) : Boolean`

Visibility: default

Description: `S_ISLNK` checks the file mode `m` to see whether the file is a symbolic link. If so it returns `True`

See also: FStat (1019), S_ISFIFO (1055), S_ISREG (1056), S_ISCHR (1055), S_ISBLK (1054), S_ISDIR (1055), S_ISSOCK (1056)

Listing: ./olinuxex/ex53.pp

```
Program Example53;

{ Program to demonstrate the S_ISLNK function. }

Uses oldlinux;

Var Info : Stat;

begin
  if LStat (paramstr(1),info) then
    begin
    if S_ISLNK(info.mode) then
      Writeln ('File is a link');
    if S_ISREG(info.mode) then
      Writeln ('File is a regular file');
    if S_ISDIR(info.mode) then
      Writeln ('File is a directory');
    if S_ISCHR(info.mode) then
      Writeln ('File is a character device file');
    if S_ISBLK(info.mode) then
      Writeln ('File is a block device file');
    if S_ISFIFO(info.mode) then
      Writeln ('File is a named pipe (FIFO)');
    if S_ISSOCK(info.mode) then
      Writeln ('File is a socket');
    end;
end.
```

23.12.110 S_ISREG

Synopsis: Is file a regular file

Declaration: `function S_ISREG(m: Word) : Boolean`

Visibility: default

Description: `S_ISREG` checks the file mode `m` to see whether the file is a regular file. If so it returns `True`

See also: FStat (1019), S_ISFIFO (1055), S_ISLNK (1055), S_ISCHR (1055), S_ISBLK (1054), S_ISDIR (1055), S_ISSOCK (1056)

23.12.111 S_ISSOCK

Synopsis: Is file a unix socket

Declaration: `function S_ISSOCK(m: Word) : Boolean`

Visibility: default

Description: `S_ISSOCK` checks the file mode `m` to see whether the file is a socket. If so it returns `True`.

See also: FStat (1019), S_ISFIFO (1055), S_ISLNK (1055), S_ISCHR (1055), S_ISBLK (1054), S_ISDIR (1055), S_ISREG (1056)

23.12.112 TCDrain

Synopsis: Terminal control: Wait till all data was transmitted

Declaration: `function TCDrain(fd: LongInt) : Boolean`

Visibility: default

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: TCFlow (1057), TCFlush (1057), TCGetAttr (1058), TCGetPGrp (1058), TCSendBreak (1059), TCSetAttr (1059), TCSetPGrp (1060), TTYName (1060), IsATTY (1031)

23.12.113 TCFlow

Synopsis: Terminal control: Suspend transmission of data

Declaration: `function TCFlow(fd: LongInt;act: LongInt) : Boolean`

Visibility: default

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: TCDrain (1057), TCFlush (1057), TCGetAttr (1058), TCGetPGrp (1058), TCSendBreak (1059), TCSetAttr (1059), TCSetPGrp (1060), TTYName (1060), IsATTY (1031)

23.12.114 TCFlush

Synopsis: Terminal control: Discard data buffer

Declaration: `function TCFlush(fd: LongInt;qsel: LongInt) : Boolean`

Visibility: default

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 buffer

TCOFLUSH output buffer

TCIOFLUSH both input and output buffers

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

Errors: Errors are reported in LinuxError.

See also: TCDrain (1057), TCFlow (1057), TCGetAttr (1058), TCGetPGrp (1058), TCSendBreak (1059), TCSetAttr (1059), TCSetPGrp (1060), TTYName (1060), IsATTY (1031)

23.12.115 TCGetAttr

Synopsis: Terminal Control: Get terminal attributes

Declaration: `function TCGetAttr(fd: LongInt;var tios: Termios) : Boolean`

Visibility: default

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: TCDrain (1057), TCFlow (1057), TCFlush (1057), TCGetPGrp (1058), TCSendBreak (1059), TCSetAttr (1059), TCSetPGrp (1060), TTYName (1060), IsATTY (1031)

Listing: ./olinuxex/ex55.pp

```
Program Example55;

uses oldlinux;

{ 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;
  WriteIn('Setting raw terminal mode');
  CFMakeRaw(tios);
  TCSetAttr(1,TCSANOW,tios);
  WriteLn('Current attributes:');
  TCGetAttr(1,tios);
  ShowTermios(tios);
  TCSetAttr(1,TCSANOW,oldios);
end.
```

23.12.116 TCGetPGrp

Synopsis: Terminal control: Get process group

Declaration: `function TCGetPGrp(fd: LongInt;var id: LongInt) : Boolean`

Visibility: default

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: TCDrain (1057), TCFlow (1057), TCFlush (1057), TCGetAttr (1058), TCSendBreak (1059), TCSetAttr (1059), TCSetPGrp (1060), TTYName (1060), IsATTY (1031)

23.12.117 TCSendBreak

Synopsis: Terminal control: Send break

Declaration:
```
function TCSendBreak(fd: LongInt;duration: LongInt) : Boolean
```

Visibility: default

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: TCDrain (1057), TCFlow (1057), TCFlush (1057), TCGetAttr (1058), TCGetPGrp (1058), TCSetAttr (1059), TCSetPGrp (1060), TTYName (1060), IsATTY (1031)

23.12.118 TCSetAttr

Synopsis: Terminal control: Set attributes

Declaration:
```
function TCSetAttr(fd: LongInt;OptAct: LongInt;const tios: Termios)
                         : Boolean
```

Visibility: default

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:

TCSANOWset immediately.

TCSADRAINwait for output.

TCSAFLUSHwait for output and discard all input not yet read.

The function Returns `True` if the call was succesfull, `False` otherwise.

For an example, see TCGetAttr (1058).

Errors: Errors are reported in LinuxError.

See also: TCDrain (1057), TCFlow (1057), TCFlush (1057), TCGetAttr (1058), TCGetPGrp (1058), TCSendBreak (1059), TCSetPGrp (1060), TTYName (1060), IsATTY (1031)

23.12.119 TCSetPGrp

Synopsis: Terminal control: Set process group

Declaration:
```
function TCSetPGrp(fd: LongInt;id: LongInt) : Boolean
```

Visibility: default

Description: `TCSetPGrp` Sets the Process Group Id to `Id`. The function returns `True` if the call was successful, `False` otherwise.

For an example, see TCGetPGrp (1058).

Errors: Errors are returned in LinuxError.

See also: TCDrain (1057), TCFlow (1057), TCFlush (1057), TCGetAttr (1058), TCGetPGrp (1058), TCSendBreak (1059), TCSetAttr (1059), TTYName (1060), IsATTY (1031)

23.12.120 TellDir

Synopsis: Return current location in a directory

Declaration:
```
function TellDir(p: PDir) : LongInt
```

Visibility: default

Description: `TellDir` returns the current location in the directory structure pointed to by `p`. It returns -1 on failure.

For an example, see OpenDir (1039).

Errors: Errors are returned in LinuxError.

See also: CloseDir (999), ReadDir (1041), SeekDir (1043), OpenDir (1039)

23.12.121 TTYname

Synopsis: Terminal control: Get terminal name

Declaration:
```
function TTYname(Handle: LongInt) : String
function TTYname(var F: Text) : String
```

Visibility: default

Description: `TTYName`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: TCDrain (1057), TCFlow (1057), TCFlush (1057), TCGetAttr (1058), TCGetPGrp (1058), TCSendBreak (1059), TCSetAttr (1059), TCSetPGrp (1060), IsATTY (1031), IOCtl (1029)

23.12.122 Umask

Synopsis: Set file creation mask.

Declaration: `function Umask(Mask: Integer) : Integer`

Visibility: default

Description: Change the file creation mask for the current user to `Mask`. The current mask is returned.

See also: Chmod (995)

Listing: ./olinuxex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Umask function. }

Uses oldlinux;

begin
  WriteIn ('Old Umask was : ',Umask(Octal(111)));
  WRiteln ('New Umask is  : ',Octal(111));
end.
```

23.12.123 Uname

Synopsis: Return system name.

Declaration: `function Uname(var unamerec: utsname) : Boolean`

Visibility: default

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 (1024), GetDomainName (1021)

23.12.124 UnLink

Synopsis: Unlink (i.e. remove) a file.

Declaration:
```
function UnLink(Path: PathStr) : Boolean
function UnLink(Path: pchar) : Boolean
```

Visibility: default

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.

For an example, see Link (1031).

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 (1031), SymLink (1052)

23.12.125 Utime

Synopsis: Set access and modification times of a file (touch).

Declaration: `function Utime(const path: PathStr;utim: UTimeBuf) : Boolean`

Visibility: default

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 (1022), Chown (996), Access (989)

Listing: ./olinuxex/ex25.pp

```
Program Example25;

{ Program to demonstrate the UTime function. }

Uses oldlinux;

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,' ');
```

```
    WriteIn (Day:2,'/',month:2,'/',year:4);
    end;
end.
```

23.12.126 WaitPid

Synopsis: Wait for a process to terminate

Declaration:
```
function WaitPid(Pid: LongInt;Status: pointer;Options: LongInt)
                        : LongInt
```

Visibility: default

Description: `WaitPid` waits for a child process with process ID `Pid` to exit. The value of `Pid` can be one of the following:

Pid < -1 Causes `WaitPid` to wait for any child process whose process group ID equals the absolute value of `pid`.

Pid = -1 Causes `WaitPid` to wait for any child process.

Pid = 0 Causes `WaitPid` to wait for any child process whose process group ID equals the one of the calling process.

Pid > 0 Causes `WaitPid` to wait for the child whose process ID equals the value of `Pid`.

The `Options` parameter can be used to specify further how `WaitPid` behaves:

WNOHANG Causes `Waitpid` to return immediately if no child hasexited.

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

__WCLONE Causes `WaitPid` also to wait for threads created by the Clone (997) call.

Upon return, it returns the exit status of the process, or -1 in case of failure.

For an example, see Fork (1016).

Errors: Errors are returned in LinuxError.

See also: Fork (1016), Execve (1006)

23.12.127 WaitProcess

Synopsis: Wait for process to terminate.

Declaration:
```
function WaitProcess(Pid: LongInt) : LongInt
```

Visibility: default

Description: `WaitProcess` waits for process `PID` to exit. `WaitProcess` is equivalent to the WaitPID (1063) call:

```
WaitPid(PID,@result,0)
```

Handles of Signal interrupts (errno=EINTR), and returns the Exitcode of Process `PID` (>=0) or -Status if it was terminated

Errors: None.

See also: WaitPID (1063), WTERMSIG (1065), WSTOPSIG (1065), WIFEXITED (1064), WIFSTOPPED (1064), WIFSIGNALED (1064), W_EXITCODE (1065), W_STOPCODE (1065), WEXITSTATUS (1064)

23.12.128 WEXITSTATUS

Synopsis: Extract the exit status from the WaitPID (1063) result.

Declaration: `function WEXITSTATUS(Status: Integer) : Integer`

Visibility: default

Description: `WEXITSTATUS` can be used to extract the exit status from `Status`, the result of the WaitPID (1063) call.

See also: WaitPID (1063), WaitProcess (1063), WTERMSIG (1065), WSTOPSIG (1065), WIFEXITED (1064), WIFSTOPPED (1064), WIFSIGNALED (1064), W_EXITCODE (1065), W_STOPCODE (1065)

23.12.129 WIFEXITED

Synopsis: Check whether the process exited normally

Declaration: `function WIFEXITED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFEXITED` checks `Status` and returns `True` if the status indicates that the process terminated normally, i.e. was not stopped by a signal.

See also: WaitPID (1063), WaitProcess (1063), WTERMSIG (1065), WSTOPSIG (1065), WIFSTOPPED (1064), WIFSIGNALED (1064), W_EXITCODE (1065), W_STOPCODE (1065), WEXITSTATUS (1064)

23.12.130 WIFSIGNALED

Synopsis: Check whether the process was exited by a signal.

Declaration: `function WIFSIGNALED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFSIGNALED` returns `True` if `Status` indicates that the process exited because it received a signal.

See also: WaitPID (1063), WaitProcess (1063), WTERMSIG (1065), WSTOPSIG (1065), WIFEXITED (1064), WIFSTOPPED (1064), W_EXITCODE (1065), W_STOPCODE (1065), WEXITSTATUS (1064)

23.12.131 WIFSTOPPED

Synopsis: Check whether the process is currently stopped.

Declaration: `function WIFSTOPPED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFSTOPPED` checks `Status` and returns `true` if the process is currently stopped. This is only possible if WUNTRACED was specified in the options of WaitPID (1063).

See also: WaitPID (1063), WaitProcess (1063), WTERMSIG (1065), WSTOPSIG (1065), WIFEXITED (1064), WIFSIGNALED (1064), W_EXITCODE (1065), W_STOPCODE (1065), WEXITSTATUS (1064)

23.12.132 WSTOPSIG

Synopsis: Return the exit code from the process.

Declaration: `function WSTOPSIG(Status: Integer) : Integer`

Visibility: default

Description: `WSTOPSIG` is an alias for WEXITSTATUS (1064).

See also: WaitPID (1063), WaitProcess (1063), WTERMSIG (1065), WIFEXITED (1064), WIFSTOPPED (1064), WIFSIGNALED (1064), W_EXITCODE (1065), W_STOPCODE (1065), WEXITSTATUS (1064)

23.12.133 WTERMSIG

Synopsis: Return the signal that caused a process to exit.

Declaration: `function WTERMSIG(Status: Integer) : Integer`

Visibility: default

Description: `WTERMSIG` extracts from `Status` the signal number which caused the process to exit.

See also: WaitPID (1063), WaitProcess (1063), WSTOPSIG (1065), WIFEXITED (1064), WIFSTOPPED (1064), WIFSIGNALED (1064), W_EXITCODE (1065), W_STOPCODE (1065), WEXITSTATUS (1064)

23.12.134 W_EXITCODE

Synopsis: Construct an exit status based on an return code and signal.

Declaration: `function W_EXITCODE(ReturnCode: Integer;Signal: Integer) : Integer`

Visibility: default

Description: `W_EXITCODE` combines `ReturnCode` and `Signal` to a status code fit for `WaitPid`.

See also: WaitPID (1063), WaitProcess (1063), WTERMSIG (1065), WSTOPSIG (1065), WIFEXITED (1064), WIFSTOPPED (1064), WIFSIGNALED (1064), W_STOPCODE (1065), WEXITSTATUS (1064)

23.12.135 W_STOPCODE

Synopsis: Construct an exit status based on a signal.

Declaration: `function W_STOPCODE(Signal: Integer) : Integer`

Visibility: default

Description: `W_STOPCODE` constructs an exit status based on `Signal`, which will cause WIFSIGNALED (1064) to return `True`

See also: WaitPID (1063), WaitProcess (1063), WTERMSIG (1065), WSTOPSIG (1065), WIFEXITED (1064), WIFSTOPPED (1064), WIFSIGNALED (1064), W_EXITCODE (1065), WEXITSTATUS (1064)

Chapter 24

Reference for unit 'ports'

24.1 Overview

The ports unit implements the `port` constructs found in Turbo Pascal. It uses classes and default array properties to do this.

The unit exists on linux, os/2 and dos. It is implemented only for compatibility with Turbo Pascal. It's usage is discouraged, because using ports is not portable programming, and the operating system may not even allow it (for instance Windows).

Under linux, your program must be run as root, or the `IOPerm` call must be set in order to set appropriate permissions on the port access.

24.2 Constants, types and variables

24.2.1 Variables

`port : tport`

Default instance of type TPort (1067). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
port[221]:=12;
```

Will result in the integer 12 being written to port 221, if port is defined as a variable of type `tport`

`portb : tport`

Default instance of type TPort (1067). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portb[221]:=12;
```

Will result in the byte 12 being written to port 221, if port is defined as a variable of type `tport`

```
portl : tportl
```

Default instance of type TPortL (1067). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portl[221]:=12;
```

Will result in the longint 12 being written to port 221, if port is defined as a variable of type `tport`

```
portw : tportw
```

Default instance of type TPortW (1068). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portw[221]:=12;
```

Will result in the word 12 being written to port 221, if port is defined as a variable of type `tport`

24.3 tport

24.3.1 Description

The `TPort` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

24.3.2 Property overview

Page	Property	Access	Description
1067	pp	rw	Access integer-sized port by port number

24.3.3 tport.pp

Synopsis: Access integer-sized port by port number

Declaration: `Property pp[w: LongInt]: Byte; default`

Visibility: public

Access: Read,Write

Description: Access integer-sized port by port number

24.4 tportl

24.4.1 Description

The `TPortL` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

24.4.2 Property overview

Page	Property	Access	Description
1068	pp	rw	Access Longint-sized port by port number

24.4.3 tportl.pp

Synopsis: Access Longint-sized port by port number

Declaration: `Property pp[w: LongInt]: LongInt; default`

Visibility: public

Access: Read,Write

Description: Access Longint-sized port by port number

24.5 tportw

24.5.1 Description

The `TPortW` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

24.5.2 Property overview

Page	Property	Access	Description
1068	pp	rw	Access word-sized port by port number

24.5.3 tportw.pp

Synopsis: Access word-sized port by port number

Declaration: `Property pp[w: LongInt]: Word; default`

Visibility: public

Access: Read,Write

Description: Access word-sized port by port number

Chapter 25

Reference for unit 'printer'

25.1 Overview

This chapter describes the PRINTER unit for Free Pascal. It was written for dos by Florian Klaempfl, and it was written for linux by Michael Van Canneyt, and has been ported to Windows and os/2 as well. Its basic functionality is the same for al supported systems, although there are minor differences on linux/unix.

25.2 Constants, types and variables

25.2.1 Variables

```
Lst : text
```

`Lst` is the standard printing device.
On linux, `Lst` is set up using `AssignLst('/tmp/PID.lst')`.

25.3 Procedures and functions

25.3.1 AssignLst

Synopsis: Assign text file to printing device

Declaration: `procedure AssignLst(var F: text;ToFile: String)`

Visibility: default

Description: `AssignLst` Assigns to `F` a printing device - *Unix only*. `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)

See also: lst (1069)

Listing: ./printex/printex.pp

```
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.');
  {$ifdef Unix}
  writeln ('Writing to pipe...');
  assignlst (f,'|/usr/bin/lpr -m');
  rewrite (f);
  for i:=1 to 80 do writeln (f,'This is line ',i,'.'#13);
  close (f);
  writeln ('Done.')
  {$endif}
end.
```

25.3.2 InitPrinter

Synopsis: Initialize the printer

Declaration: `procedure InitPrinter(const PrinterName: String)`

Visibility: default

Description: Initialize the printer

25.3.3 IsLstAvailable

Synopsis: Determine whether printer is available.

Declaration: `function IsLstAvailable : Boolean`

Visibility: default

Description: Determine whether printer is available.

Chapter 26

Reference for unit 'Sockets'

26.1 Used units

Table 26.1: Used units by unit 'Sockets'

Name	Page
baseunix	1071
UnixType	1071

26.2 Overview

This document describes the SOCKETS unit for Free Pascal. it was written for linux by Michael Van Canneyt, and ported to Windows by Florian Klaempfl.

26.3 Constants, types and variables

26.3.1 Constants

```
AF_APPLETALK = 5
```

Address family Appletalk DDP

```
AF_ASH = 18
```

Address family: Ash

```
AF_ATMPVC = 8
```

Address family: ATM PVCs

```
AF_ATMSVC = 20
```

Address family: ATM SVCs

```
AF_AX25 = 3
```

Address family Amateur Radio AX.25

```
AF_BLUETOOTH = 31
```

Address family: Bluetooth sockets

```
AF_BRIDGE = 7
```

Address family Multiprotocol bridge

```
AF_DECnet = 12
```

Address family: Reserved for DECnet project.

```
AF_ECONET = 19
```

Address family: Acorn Econet

```
AF_INET = 2
```

Address family Internet IP Protocol

```
AF_INET6 = 10
```

Address family IP version 6

```
AF_IPX = 4
```

Address family Novell IPX

```
AF_IRDA = 23
```

Address family: IRDA sockets

```
AF_KEY = 15
```

Address family: PF_KEY key management API

```
AF_LLC = 26
```

Address family: Linux LLC

```
AF_LOCAL = 1
```

Address family: Unix socket

```
AF_MAX = 32
```

Address family Maximum value

```
AF_NETBEUI = 13
```

Address family: Reserved for 802.2LLC project

```
AF_NETLINK = 16
```

Address family: ?

```
AF_NETROM = 6
```

Address family Amateur radio NetROM

```
AF_PACKET = 17
```

Address family: Packet family

```
AF_PPPOX = 24
```

Address family: PPPoX sockets

```
AF_ROSE = 11
```

Address family: Amateur Radio X.25 PLP

```
AF_ROUTE = AF_NETLINK
```

Address family: Alias to emulate 4.4BSD.

```
AF_SECURITY = 14
```

Address family: Security callback pseudo AF

```
AF_SNA = 22
```

Addresss family: Linux SNA project

```
AF_TIPC = 30
```

Address family: TIPC sockets

```
AF_UNIX = 1
```

Address family Unix domain sockets

```
AF_UNSPEC = 0
```

Address family Not specified

```
AF_WANPIPE = 25
```

Address family: Wanpipe API Sockets

```
AF_X25 = 9
```

Address family Reserved for X.25 project

```
EsockEACCESS = ESysEAcces
```

Access forbidden error

```
EsockEBADF = EsysEBADF
```

Alias: bad file descriptor

```
EsockEFAULT = EsysEFAULT
```

Alias: an error occurred

```
EsockEINTR = EsysEINTR
```

Alias : operation interrupted

```
EsockEINVAL = EsysEINVAL
```

Alias: Invalid value specified

```
EsockEMFILE = ESysEmfile
```

Error code ?

```
EsockEMSGSIZE = ESysEMsgSize
```

Wrong message size error

```
EsockENOBUFS = ESysENoBufs
```

No buffer space available error

```
EsockENOTCONN = ESysENotConn
```

Not connected error

```
EsockENOTSOCK = ESysENotSock
```

File descriptor is not a socket error

```
EsockEPROTONOSUPPORT = ESysEProtoNoSupport
```

Protocol not supported error

```
EsockEWOULDBLOCK = ESysEWouldBlock
```

Operation would block error

```
INADDR_ANY = CARDINAL ( 0 )
```

Undocumented ?

```
INADDR_NONE = CARDINAL ( $FFFFFFFF )
```

Undocumented ?

```
IPPROTO_AH = 51
```

authentication header.

```
IPPROTO_COMP = 108
```

Compression Header Protocol.

```
IPPROTO_DSTOPTS = 60
```

IPv6 destination options.

```
IPPROTO_EGP = 8
```

Exterior Gateway Protocol.

```
IPPROTO_ENCAP = 98
```

Encapsulation Header.

```
IPPROTO_ESP = 50
```

encapsulating security payload.

```
IPPROTO_FRAGMENT = 44
```

IPv6 fragmentation header.

```
IPPROTO_GRE = 47
```

General Routing Encapsulation.

```
IPPROTO_HOPOPTS = 0
```

IPv6 Hop-by-Hop options.

```
IPPROTO_ICMP = 1
```

Internet Control Message Protocol.

```
IPPROTO_ICMPV6 = 58
```

ICMPv6.

```
IPPROTO_IDP = 22
```

XNS IDP protocol.

```
IPPROTO_IGMP = 2
```

Internet Group Management Protocol.

```
IPPROTO_IP = 0
```

Dummy protocol for TCP.

```
IPPROTO_IPIP = 4
```

IPIP tunnels (older KA9Q tunnels use 94).

```
IPPROTO_IPV6 = 41
```

IPv6 header.

```
IPPROTO_MAX = 255
```

Maximum value for IPPROTO options

```
IPPROTO_MTP = 92
```

Multicast Transport Protocol.

```
IPPROTO_NONE = 59
```

IPv6 no next header.

```
IPPROTO_PIM = 103
```

Protocol Independent Multicast.

```
IPPROTO_PUP = 12
```

PUP protocol.

```
IPPROTO_RAW = 255
```

Raw IP packets.

```
IPPROTO_ROUTING = 43
```

IPv6 routing header.

```
IPPROTO_RSVP = 46
```

Reservation Protocol.

```
IPPROTO_SCTP = 132
```

Stream Control Transmission Protocol.

```
IPPROTO_TCP = 6
```

Transmission Control Protocol.

```
IPPROTO_TP = 29
```

SO Transport Protocol Class 4.

```
IPPROTO_UDP = 17
```

User Datagram Protocol.

```
IPV6_ADDRFORM = 1
```

Undocumented Getsockopt option ?

```
IPV6_ADD_MEMBERSHIP = IPV6_JOIN_GROUP
```

Undocumented Getsockopt option ?

```
IPV6_AUTHHDR = 10
```

Undocumented Getsockopt option ?

```
IPV6_CHECKSUM = 7
```

Undocumented Getsockopt option ?

```
IPV6_DROP_MEMBERSHIP = IPV6_LEAVE_GROUP
```

Undocumented Getsockopt option ?

```
IPV6_DSTOPTS = 4
```

Undocumented Getsockopt option ?

```
IPV6_HOPLIMIT = 8
```

Undocumented Getsockopt option ?

```
IPV6_HOPOPTS = 3
```

Undocumented Getsockopt option ?

```
IPV6_IPSEC_POLICY = 34
```

Undocumented Getsockopt option ?

```
IPV6_JOIN_ANYCAST = 27
```

Undocumented Getsockopt option ?

```
IPV6_JOIN_GROUP = 20
```

Undocumented Getsockopt option ?

```
IPV6_LEAVE_ANYCAST = 28
```

Undocumented Getsockopt option ?

```
IPV6_LEAVE_GROUP = 21
```

Undocumented Getsockopt option ?

```
IPV6_MTU = 24
```

Undocumented Getsockopt option ?

```
IPV6_MTU_DISCOVER = 23
```

Undocumented Getsockopt option ?

```
IPV6_MULTICAST_HOPS = 18
```

Undocumented Getsockopt option ?

```
IPV6_MULTICAST_IF = 17
```

Undocumented Getsockopt option ?

```
IPV6_MULTICAST_LOOP = 19
```

Undocumented Getsockopt option ?

```
IPV6_NEXTHOP = 9
```

Undocumented Getsockopt option ?

```
IPV6_PKTINFO = 2
```

Undocumented Getsockopt option ?

```
IPV6_PKTOPTIONS = 6
```

Undocumented Getsockopt option ?

```
IPV6_PMTUDISC_DO = 2
```

Always DF.

```
IPV6_PMTUDISC_DONT = 0
```

Never send DF frames.

```
IPV6_PMTUDISC_WANT = 1
```

Use per route hints.

```
IPV6_RECVERR = 25
```

Undocumented Getsockopt option ?

```
IPV6_ROUTER_ALERT = 22
```

Undocumented Getsockopt option ?

```
IPV6_RTHDR = 5
```

Undocumented Getsockopt option ?

```
IPV6_RTHDR_LOOSE = 0
```

Hop doesn't need to be neighbour.

```
IPV6_RTHDR_STRICT = 1
```

Hop must be a neighbour.

```
IPV6_RTHDR_TYPE_0 = 0
```

IPv6 Routing header type 0.

```
IPV6_RXDSTOPTS = IPV6_DSTOPTS
```

Undocumented Getsockopt option ?

```
IPV6_RXHOPOPTS = IPV6_HOPOPTS
```

Undocumented Getsockopt option ?

```
IPV6_RXSRCRT = IPV6_RTHDR
```

Undocumented Getsockopt option ?

```
IPV6_UNICAST_HOPS = 16
```

Undocumented Getsockopt option ?

```
IPV6_V6ONLY = 26
```

Undocumented Getsockopt option ?

```
IPV6_XFRM_POLICY = 35
```

Undocumented Getsockopt option ?

```
IP_ADD_MEMBERSHIP = 35
```

add an IP group membership

```
IP_ADD_SOURCE_MEMBERSHIP = 39
```

join source group

```
IP_BLOCK_SOURCE = 38
```

block data from source

```
IP_DEFAULT_MULTICAST_LOOP = 1
```

Undocumented ?

```
IP_DEFAULT_MULTICAST_TTL = 1
```

Undocumented ?

```
IP_DROP_MEMBERSHIP = 36
```

drop an IP group membership

```
IP_DROP_SOURCE_MEMBERSHIP = 40
```

leave source group

```
IP_HDRINCL = 3
```

Header is included with data.

```
IP_MAX_MEMBERSHIPS = 20
```

Undocumented ?

```
IP_MSFILTER = 41
```

Undocumented ?

```
IP_MTU_DISCOVER = 10
```

Undocumented ?

```
IP_MULTICAST_IF = 32
```

set/get IP multicast i/f

```
IP_MULTICAST_LOOP = 34
```

set/get IP multicast loopback

```
IP_MULTICAST_TTL = 33
```

set/get IP multicast ttl

```
IP_OPTIONS = 4
```

IP per-packet options.

```
IP_PKTINFO = 8
```

Undocumented ?

```
IP_PKTOPTIONS = 9
```

Undocumented ?

```
IP_PMTUDISC = 10
```

Undocumented ?

```
IP_PMTUDISC_DO = 2
```

Always DF.

```
IP_PMTUDISC_DONT = 0
```

Never send DF frames.

```
IP_PMTUDISC_WANT = 1
```

Use per route hints.

```
IP_RECVERR = 11
```

Undocumented ?

```
IP_RECVOPTS = 6
```

Receive all IP options w/datagram.

```
IP_RECVRETOPTS = IP_RETOPTS
```

Receive IP options for response.

```
IP_RECVTOS = 13
```

Undocumented ?

```
IP_RECVTTL = 12
```

Undocumented ?

```
IP_RETOPTS = 7
```

Set/get IP per-packet options.

```
IP_ROUTER_ALERT = 5
```

Undocumented ?

```
IP_TOS = 1
```

IP type of service and precedence.

```
IP_TTL = 2
```

IP time to live.

```
IP_UNBLOCK_SOURCE = 37
```

unblock data from source

```
MCAST_BLOCK_SOURCE = 43
```

block from given group

```
MCAST_EXCLUDE = 0
```

Undocumented ?

```
MCAST_INCLUDE = 1
```

Undocumented ?

```
MCAST_JOIN_GROUP = 42
```

join any-source group

```
MCAST_JOIN_SOURCE_GROUP = 46
```

join source-spec gruoup

```
MCAST_LEAVE_GROUP = 45
```

leave any-source group

```
MCAST_LEAVE_SOURCE_GROUP = 47
```

leave source-spec group

```
MCAST_MSFILTER = 48
```

Undocumented ?

```
MCAST_UNBLOCK_SOURCE = 44
```

unblock from given group

```
MSG_CONFIRM = $0800
```

Send flags: Conform connection

```
MSG_CTRUNC = $0008
```

Receive flags: Control Data was discarded (buffer too small)

```
MSG_DONTROUTE = $0004
```

Send flags: don't use gateway

```
MSG_DONTWAIT = $0040
```

Receive flags: Non-blocking operation request.

```
MSG_EOF = MSG_FIN
```

Alias for `MSG_FIN`

```
MSG_EOR = $0080
```

Receive flags: End of record

```
MSG_ERRQUERE = $2000
```

Receive flags: ?

```
MSG_FIN = $0200
```

Receive flags: ?

```
MSG_MORE = $8000
```

Receive flags: ?

```
MSG_NOSIGNAL = $4000
```

Receive flags: Suppress SIG_PIPE signal.

```
MSG_OOB = $0001
```

Receive flags: receive out-of-band data.

```
MSG_PEEK = $0002
```

Receive flags: peek at data, don't remove from buffer.

```
MSG_PROXY = $0010
```

Receive flags: ?

```
MSG_RST = $1000
```

Receive flags: ?

```
MSG_SYN = $0400
```

Receive flags: ?

```
MSG_TRUNC = $0020
```

Receive flags: packet Data was discarded (buffer too small)

```
MSG_TRYHARD = MSG_DONTROUTE
```

Receive flags: ?

```
MSG_WAITALL = $0100
```

Receive flags: Wait till operation completed.

```
NoAddress : in_addr = (s_addr:0 )
```

Constant indicating invalid (no) network address.

```
NoAddress6 : in6_addr = (u6_addr16: ( 0,0,0,0,0,0,0,0 ) )
```

Constant indicating invalid (no) IPV6 network address.

```
NoNet : in_addr = (s_addr:0 )
```

Constant indicating invalid (no) network address.

```
NoNet6 : in6_addr = (u6_addr16: ( 0,0,0,0,0,0,0,0 ) )
```

Constant indicating invalid (no) IPV6 network address.

```
PF_APPLETALK = AF_APPLETALK
```

Protocol family: Appletalk DDP

```
PF_ASH = AF_ASH
```

Protocol family: Ash

```
PF_ATMPVC = AF_ATMPVC
```

Protocol family: ATM PVCs

```
PF_ATMSVC = AF_ATMSVC
```

Protocol family: ATM SVCs

```
PF_AX25 = AF_AX25
```

Protocol family: Amateur Radio AX.25

```
PF_BLUETOOTH = AF_BLUETOOTH
```

Protocol family: Bluetooth sockets

```
PF_BRIDGE = AF_BRIDGE
```

Protocol family: Multiprotocol bridge

```
PF_DECnet = AF_DECnet
```

Protocol Family: DECNET project

```
PF_ECONET = AF_ECONET
```

Protocol family: Acorn Econet

```
PF_INET = AF_INET
```

Protocol family: Internet IP Protocol

```
PF_INET6 = AF_INET6
```

Protocol family: IP version 6

```
PF_IPX = AF_IPX
```

Protocol family: Novell IPX

```
PF_IRDA = AF_IRDA
```

Protocol family: IRDA sockets

```
PF_KEY = AF_KEY
```

Protocol family: Key management API

```
PF_LLC = AF_LLC
```

Protocol family: Linux LLC

```
PF_LOCAL = AF_LOCAL
```

Protocol family: Unix socket

```
PF_MAX = AF_MAX
```

Protocol family: Maximum value

```
PF_NETBEUI = AF_NETBEUI
```

Protocol family: Reserved for 802.2LLC project

```
PF_NETLINK = AF_NETLINK
```

Protocol family: ?

```
PF_NETROM = AF_NETROM
```

Protocol family:Amateur radio NetROM

```
PF_PACKET = AF_PACKET
```

Protocol family: Packet family

```
PF_PPPOX = AF_PPPOX
```

Protocol family: PPPoX sockets

```
PF_ROSE = AF_ROSE
```

Protocol family: Amateur Radio X.25 PLP

```
PF_ROUTE = AF_ROUTE
```

Protocol Family: ?

```
PF_SECURITY = AF_SECURITY
```

Protocol family: Security callback pseudo PF

```
PF_SNA = AF_SNA
```

Protocol Family: Linux SNA project

```
PF_TIPC = AF_TIPC
```

Protocol family: TIPC sockets

```
PF_UNIX = AF_UNIX
```

Protocol family: Unix domain sockets

```
PF_UNSPEC = AF_UNSPEC
```

Protocol family: Unspecified

```
PF_WANPIPE = AF_WANPIPE
```

Protocol family: Wanpipe API Sockets

```
PF_X25 = AF_X25
```

Protocol family: Reserved for X.25 project

```
SCM_SRCRT = IPV6_RXSRCRT
```

Undocumented Getsockopt option ?

```
SCM_TIMESTAMP = SO_TIMESTAMP
```

Socket option: ?

```
SHUT_RD = 0
```

Shutdown read part of full duplex socket

```
SHUT_RDWR = 2
```

Shutdown read and write part of full duplex socket

```
SHUT_WR = 1
```

Shutdown write part of full duplex socket

```
SOCK_DGRAM = 2
```

Type of socket: datagram (conn.less) socket (UDP)

```
SOCK_MAXADDRLEN = 255
```

Maximum socket address length for Bind (1095) call.

```
SOCK_RAW = 3
```

Type of socket: raw socket

```
SOCK_RDM = 4
```

Type of socket: reliably-delivered message

```
SOCK_SEQPACKET = 5
```

Type of socket: sequential packet socket

```
SOCK_STREAM = 1
```

Type of socket: stream (connection) type socket (TCP)

```
SOL_ICMPV6 = 58
```

Socket level values for IPv6: ICMPV6

```
SOL_IP = 0
```

Undocumented ?

```
SOL_IPV6 = 41
```

Socket level values for IPv6: IPV6

```
SOL_SOCKET = 1
```

Socket option level: Socket level

```
SOMAXCONN = 128
```

Maximum queue length specifiable by listen.

```
SO_ACCEPTCONN = 30
```

Socket option: ?

```
SO_ATTACH_FILTER = 26
```

Socket option: ?

```
SO_BINDTODEVICE = 25
```

Socket option: ?

```
SO_BROADCAST = 6
```

Socket option: Broadcast

```
SO_BSDCOMPAT = 14
```

Socket option: ?

```
SO_DEBUG = 1
```

Socket option level: debug

```
SO_DETACH_FILTER = 27
```

Socket option: ?

```
SO_DONTROUTE = 5
```

Socket option: Don't route

```
SO_ERROR = 4
```

Socket option: Error

```
SO_KEEPALIVE = 9
```

Socket option: keep alive

```
SO_LINGER = 13
```

Socket option: ?

```
SO_NO_CHECK = 11
```

Socket option: ?

```
SO_OOBINLINE = 10
```

Socket option: ?

```
SO_PASSCRED = 16
```

Socket option: ?

```
SO_PEERCRED = 17
```

Socket option: ?

```
SO_PEERNAME = 28
```

Socket option: ?

```
SO_PRIORITY = 12
```

Socket option: ?

```
SO_RCVBUF = 8
```

Socket option: receive buffer

```
SO_RCVLOWAT = 18
```

Socket option: ?

```
SO_RCVTIMEO = 20
```

Socket option: ?

```
SO_REUSEADDR = 2
```

Socket option: Reuse address

```
SO_SECURITY_AUTHENTICATION = 22
```

Socket option: ?

```
SO_SECURITY_ENCRYPTION_NETWORK = 24
```

Socket option: ?

```
SO_SECURITY_ENCRYPTION_TRANSPORT = 23
```

Socket option: ?

```
SO_SNDBUF = 7
```

Socket option: Send buffer

```
SO_SNDLOWAT = 19
```

Socket option: ?

```
SO_SNDTIMEO = 21
```

Socket option: ?

```
SO_TIMESTAMP = 29
```

Socket option: ?

```
SO_TYPE = 3
```

Socket option: Type

```
S_IN = 0
```

Input socket in socket pair.

```
S_OUT = 1
```

Output socket in socket pair

26.3.2 Types

```
in6_addr = packed record
end
```

Record used to describe a general IPV6 address.

```
in_addr = packed record
end
```

General inet socket address.

```
linger = packed record
  l_onoff : cint;
  l_linger : cint;
end
```

This record is used in the setsockopt (1071) call to specify linger options.

```
PIn6Addr = pin6_addr
```

Pointer to in6_addr (1090) type.

```
pin6_addr = ^in6_addr
```

Pointer to Tin6_addr (1092)

```
PInAddr = pin_addr
```

Alias for pin_addr (1091)

```
PInetSockAddr = psockaddr_in
```

Pointer to sockaddr_in (1092)

```
PInetSockAddr6 = psockaddr_in6
```

Pointer to sockaddr_in6 (1092) type

```
pin_addr = ^in_addr
```

Pointer to in_addr (1091) record.

```
plinger = ^linger
```

Pointer to linger (1091) type.

```
psockaddr = ^sockaddr
```

Pointer to TSockAddr (1093)

```
psockaddr_in = ^sockaddr_in
```

Pointer to sockaddr_in (1092)

```
psockaddr_in6 = ^sockaddr_in6
```

Pointer to sockaddr_in6 (1092)

```
psockaddr_un = ^sockaddr_un
```

Pointer to sockaddr_un (1092) type.

```
sa_family_t = cushort
```

Address family type

```
sockaddr = packed record
end
```

`sockaddr` is used to store a general socket addres for the Bind (1095), Recv (1110) and Send (1111) calls.

```
sockaddr_in = packed record
end
```

`sockaddr_in` is used to store a INET socket addres for the Bind (1095), Recv (1110) and Send (1111) calls.

```
sockaddr_in6 = packed record
  sin6_family : sa_family_t;
  sin6_port : cuint16;
  sin6_flowinfo : cuint32;
  sin6_addr : in6_addr;
  sin6_scope_id : cuint32;
end
```

Alias for sockaddr_in6 (1092)

```
sockaddr_un = packed record
  sun_family : sa_family_t;
  sun_path : Array[0..107] of Char;
end
```

`sockaddr_un` is used to store a UNIX socket addres for the Bind (1095), Recv (1110) and Send (1111) calls.

```
TIn6Addr = in6_addr
```

Alias for in6_addr (1090) type.

```
Tin6_addr = in6_addr
```

Alias for sockaddr_in6 (1092)

```
TInAddr = in_addr
```

Alias for in_addr (1091) record type.

```
TInetSockAddr = sockaddr_in
```

Alias for sockaddr_in (1092)

```
TInetSockAddr6 = sockaddr_in6
```

Alias for sockaddr_in6 (1092)

```
TIn_addr = in_addr
```

Alias for in_addr (1091) record type.

```
TLinger = linger
```

Alias for linger (1071)

```
TSockAddr = sockaddr
```

```
TSockArray = Array[1..2] of LongInt
```

Type returned by the SocketPair (1115) call.

```
Tsocket = LongInt
```

Alias for easy kylix porting

```
TSockPairArray = Array[0..1] of LongInt
```

Array of sockets, used in SocketPair (1115) call.

```
TUnixSockAddr = packed record
  family : sa_family_t;
  path : Array[0..107] of Char;
end
```

Alias for sockaddr_un (1092)

26.4 Procedures and functions

26.4.1 Accept

Synopsis: Accept a connection from a socket (deprecated).

Declaration:
```
function Accept(Sock: LongInt;var Addr;var Addrlen: LongInt) : LongInt
function Accept(Sock: LongInt;var addr: TInetSockAddr;var SockIn: File;
                var SockOut: File) : Boolean
function Accept(Sock: LongInt;var addr: TInetSockAddr;var SockIn: text;
                var SockOut: text) : Boolean
function Accept(Sock: LongInt;var addr: String;var SockIn: text;
                var SockOut: text) : Boolean
function Accept(Sock: LongInt;var addr: String;var SockIn: File;
                var SockOut: File) : Boolean
```

Visibility: default

Description: `Accept` accepts a connection from a socket `Sock`, which was listening for a connection. If a connection is accepted, a file descriptor is returned. On error `-1` is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The `Accept` call fills the address of the connecting entity in `Addr`, and sets its length in `Addrlen`. `Addr` should be pointing to enough space, and `Addrlen` should be set to the amount of space available, prior to the call.

The alternate forms of the Accept (1093) command, with the `Text` or `File` parameters are equivalent to subsequently calling the regular Accept (1093) function and the Sock2Text (1114) or Sock2File (1114) functions. These functions return `True` if successful, `False` otherwise.

Errors: On error, `-1` is returned, and 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 (1109), Connect (1096), Bind (1095)

Listing: ./sockex/socksvr.pp

```
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 Sockets;

Var
  FromName : string;
  Buffer   : string[255];
  S        : Longint;
  Sin,Sout : Text;
  SAddr    : TInetSockAddr;

procedure perror (const S:string);
begin
  writeln (S,SocketError);
  halt(100);
```

```
end;


begin
  S:=Socket (AF_INET,SOCK_STREAM,0);
  if SocketError<>0 then
   Perror ('Server : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(50000);
  SAddr.sin_addr.s_addr:=0;
  if not Bind(S,SAddr,sizeof(saddr)) then
   PError ('Server : Bind : ');
  if not Listen (S,1) then
   PError ('Server : Listen : ');
  Writeln('Waiting for Connect from Client, run now sock_cli in an other tty');
  if not Accept (S,FromName,Sin,Sout) then
   PError ('Server : Accept : '+fromname);
  Reset(Sin);
  ReWrite(Sout);
  Writeln(Sout,'Message From Server');
  Flush(SOut);
  while not eof(sin) do
   begin
     Readln(Sin,Buffer);
     Writeln('Server : read : ',buffer);
   end;
end.
```

26.4.2 Bind

Synopsis: Bind a socket to an address (deprecated).

Declaration:
```
function Bind(Sock: LongInt;const Addr;AddrLen: LongInt) : Boolean
function Bind(Sock: LongInt;const addr: String) : Boolean
```

Visibility: default

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.

The form of the `Bind` command with the TUnixSockAddr (1093) is equivalent to subsequently calling Str2UnixSockAddr (1115) and the regular `Bind` function. The function returns `True` if successfull, `False` otherwise.

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 (1114)

26.4.3 CloseSocket

Synopsis: Closes a socket handle.

Declaration:
```
function CloseSocket(Sock: LongInt) : LongInt
```

Visibility: default

Description: `CloseSocket` closes a socket handle. It returns 0 if the socket was closed succesfully, -1 if it failed.

Errors: On error, -1 is returned.

See also: Socket (1114)

26.4.4 Connect

Synopsis: Open a connection to a server socket (deprecated).

Declaration:
```
function Connect(Sock: LongInt;const Addr;Addrlen: LongInt) : Boolean
function Connect(Sock: LongInt;const addr: TInetSockAddr;
                 var SockIn: text;var SockOut: text) : Boolean
function Connect(Sock: LongInt;const addr: TInetSockAddr;
                 var SockIn: File;var SockOut: File) : Boolean
function Connect(Sock: LongInt;const addr: String;var SockIn: text;
                 var SockOut: text) : Boolean
function Connect(Sock: LongInt;const addr: String;var SockIn: File;
                 var SockOut: File) : Boolean
```

Visibility: default

Description: `Connect` opens a connection to a peer, whose address is described by `Addr`. `AddrLen` contains the length of the address. The type of `Addr` depends on the kind of connection you're trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The forms of the Connect (1096) command with the `Text` or `File` arguments are equivalent to subsequently calling the regular `Connect` function and the Sock2Text (1114) or Sock2File (1114) functions. These functions return `True` if successfull, `False` otherwise.

The `Connect` function returns a file descriptor if the call was successfull, `-1` in case of error.

Errors: On error, `-1` is returned and errors are reported in `SocketError`.

See also: Listen (1109), Bind (1095), Accept (1093)

Listing: ./sockex/sockcli.pp

```
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;

procedure PError(const S : string);
begin
  writeln(S,SocketError);
```

```
  halt(100);
end;


Var
  SAddr     : TInetSockAddr;
  Buffer    : string [255];
  S         : Longint;
  Sin,Sout  : Text;
  i         : integer;

begin
  S:=Socket (AF_INET,SOCK_STREAM,0);
  if SocketError<>0 then
   Perror('Client : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(50000);
  { localhost : 127.0.0.1 in network order }
  SAddr.sin_addr.s_addr:=HostToNet((127 shl 24) or 1);
  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
   WriteIn(Sout,Buffer);
  Flush(Sout);
  ReadIn(SIn,Buffer);
  WriteLn(Buffer);
  Close(sout);
end.
```

Listing: ./sockex/pfinger.pp

```
program pfinger;

uses sockets,errors;

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

begin
  Addr.sin_family:=AF_INET;
  { port 79 in network order }
  Addr.sin_port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.sin_addr.s_addr:=((1 shl 24) or 127);
  S:=Socket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,ADDR,SIN,SOUT) Then
    begin
    WriteIn ('Couldn''t connect to localhost');
    WriteIn ('Socket error : ',strerror(SocketError));
    halt(1);
    end;
  rewrite (sout);
```

```
  reset(sin);
  writeln (sout,paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
    readln (Sin,line);
    writeln (line);
    end;
  Shutdown(s,2);
  close (sin);
  close (sout);
end.
```

26.4.5 fpaccept

Synopsis: Accept a connection from a socket.

Declaration: `function fpaccept(s: cint;addrx: psockaddr;addrlen: psocklen) : cint`

Visibility: default

Description: `Accept` accepts a connection from a socket `S`, which was listening for a connection. If a connection is accepted, a file descriptor is returned (positive number). On error `-1` is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The `Accept` call fills the address of the connecting entity in `Addrx`, and sets its length in `Addrlen`. `Addrx` should be pointing to enough space, and `Addrlen` should be set to the amount of space available, prior to the call.

Errors: On error, `-1` is returned, and errors are reported in `SocketError`, and include the following:

SYS_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: fpListen (1103), fpConnect (1100), fpBind (1099)

Listing: ./sockex/socksvr.pp

```
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 Sockets;

Var
  FromName : string;
  Buffer   : string[255];
  S        : Longint;
  Sin,Sout : Text;
```

```
  SAddr     : TInetSockAddr;

procedure perror (const S:string);
begin
  writeln (S,SocketError);
  halt(100);
end;



begin
  S:=Socket (AF_INET,SOCK_STREAM,0);
  if SocketError<>0 then
   Perror ('Server : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(50000);
  SAddr.sin_addr.s_addr:=0;
  if not Bind(S,SAddr,sizeof(saddr)) then
   PError ('Server : Bind : ');
  if not Listen (S,1) then
   PError ('Server : Listen : ');
  WriteIn('Waiting for Connect from Client, run now sock_cli in an other tty');
  if not Accept (S,FromName,Sin,Sout) then
   PError ('Server : Accept : '+fromname);
  Reset(Sin);
  ReWrite(Sout);
  WriteIn(Sout,'Message From Server');
  Flush(SOut);
  while not eof(sin) do
   begin
     ReadIn(Sin,Buffer);
     WriteIn('Server : read : ',buffer);
   end;
end.
```

26.4.6 fpbind

Synopsis: Bind a socket to an address.

Declaration: `function fpbind(s: cint;addrx: psockaddr;addrlen: tsocklen) : cint`

Visibility: default

Description: `fpBind` binds the socket `s` to address `Addrx`. `Addrx` has length `Addrlen`. The function returns `0` if the call was succesful, `-1` 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 (1114)

26.4.7 fpconnect

Synopsis: Open a connection to a server socket.

Declaration: `function fpconnect(s: cint;name: psockaddr;namelen: tsocklen) : cint`

Visibility: default

Description: `fpConnect` opens a connection to a peer, whose address is described by `Name`. `NameLen` contains the length of the address. The type of `Name` depends on the kind of connection you're trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The `Connect` function returns a file descriptor if the call was successfull, `-1` in case of error.

Errors: On error, `-1` is returned and errors are reported in `SocketError`.

See also: fpListen (1103), fpBind (1099), fpAccept (1098)

Listing: ./sockex/sockcli.pp

```
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;

procedure PError(const S : string);
begin
  writeln(S,SocketError);
  halt(100);
end;


Var
  SAddr     : TInetSockAddr;
  Buffer    : string [255];
  S         : Longint;
  Sin,Sout  : Text;
  i         : integer;

begin
  S:=Socket (AF_INET,SOCK_STREAM,0);
  if SocketError<>0 then
   Perror('Client : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(50000);
  { localhost : 127.0.0.1 in network order }
  SAddr.sin_addr.s_addr:=HostToNet((127 shl 24) or 1);
  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
   WriteIn(Sout,Buffer);
```

```
  Flush(Sout);
  ReadLn(SIn,Buffer);
  WriteLn(Buffer);
  Close(sout);
end.
```

Listing: ./sockex/pfinger.pp

```
program pfinger;

uses sockets,errors;

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

begin
  Addr.sin_family:=AF_INET;
  { port 79 in network order }
  Addr.sin_port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.sin_addr.s_addr:=((1 shl 24) or 127);
  S:=Socket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,ADDR,SIN,SOUT) Then
    begin
    WriteLn ('Couldn''t connect to localhost');
    WriteLn ('Socket error : ',strerror(SocketError));
    halt(1);
    end;
  rewrite (sout);
  reset(sin);
  writeln (sout,paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
    readln (Sin,line);
    writeln (line);
    end;
  Shutdown(s,2);
  close (sin);
  close (sout);
end.
```

26.4.8 fpgetpeername

Synopsis: Return the name (address) of the connected peer.

Declaration: `function fpgetpeername(s: cint;name: psockaddr;namelen: psocklen) : cint`

Visibility: default

Description: `fpGetPeerName` returns the name of the entity connected to the specified socket `S`. The Socket must be connected for this call to work.

`Name` should point to enough space to store the name, the amount of space pointed to should be set in `Namelen`. When the function returns succesfully, `Name` will be filled with the name, and `Name` will be set to the length of `Name`.

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: fpConnect (1100), fpSocket (1106)

26.4.9 fpgetsockname

Synopsis: Return name of socket.

Declaration:
```
function fpgetsockname(s: cint;name: psockaddr;namelen: psocklen) : cint
```

Visibility: default

Description: `fpGetSockName` returns the current name of the specified socket `S`. `Name` should point to enough space to store the name, the amount of space pointed to should be set in `Namelen`. When the function returns succesfully, `Name` will be filled with the name, and `Namelen` will be set to the length of `Name`.

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: fpBind (1099)

26.4.10 fpgetsockopt

Synopsis: Get current socket options

Declaration:
```
function fpgetsockopt(s: cint;level: cint;optname: cint;optval: pointer;
                      optlen: psocklen) : cint
```

Visibility: default

Description: `fpGetSockOpt` gets the connection option `optname`, for socket `S`. 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 interpret the option.

The options are stored in the memory location pointed to by `optval`. `optlen` should point to the initial length of `optval`, and on return will contain the actual length of the stored data.

On success, 0 is returned. On Error, -1 is returned.

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: fpSetSockOpt (1105)

26.4.11 fplisten

Synopsis: Listen for connections on a socket.

Declaration: `function fplisten(s: cint;backlog: cint) : cint`

Visibility: default

Description: `fpListen` listens for up to `backlog` connections from socket `S`. The socket `S` must be of type `SOCK_STREAM` or `Sock_SEQPACKET`.

The function returns `0` if a connection was accepted, `-1` 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: fpSocket (1106), fpBind (1099), fpConnect (1100)

26.4.12 fprecv

Synopsis: Receive data on socket

Declaration: `function fprecv(s: cint;buf: pointer;len: size_t;flags: cint) : ssize_t`

Visibility: default

Description: `fpRecv` reads at most `len` bytes from socket `S` into address `buf`. 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 (1111)

26.4.13 fprecvfrom

Synopsis: Receive data from an unconnected socket

Declaration:
```
function fprecvfrom(s: cint;buf: pointer;len: size_t;flags: cint;
                    from: psockaddr;fromlen: psocklen) : ssize_t
```

Visibility: default

Description: `fpRecvFrom` receives data in buffer `Buf` with maximum length `Len` from socket `S`. Receipt is controlled by options in `Flags`. The location pointed to by `from` will be filled with the address from the sender, and it's length will be stored in `fromlen`. The function returns the number of bytes received, or -1 on error. `AddrLen`.

Errors: On error, -1 is returned.

See also: fpSocket (1106), fprecv (1103)

26.4.14 fpsend

Synopsis: Send data through socket

Declaration:
```
function fpsend(s: cint;msg: pointer;len: size_t;flags: cint) : ssize_t
```

Visibility: default

Description: `fpSend` sends `Len` bytes starting from address `Msg` to socket `S`. `S` must be in a connected state. Options can be passed in `Flags`.

The function returns the number of bytes sent, or -1 if a detectable error occurred.

`Flags` can be one of the following:

1Process out-of band data.

4Bypass 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: fpRecv (1103)

26.4.15 fpsendto

Synopsis: Send data through an unconnected socket to an address.

Declaration:
```
function fpsendto(s: cint;msg: pointer;len: size_t;flags: cint;
                  tox: psockaddr;tolen: tsocklen) : ssize_t
```

Visibility: default

Description: `fpSendTo` sends data from buffer `Msg` with length `len` through socket `S` with options `Flags`. The data is sent to address `tox`, which has length `toLen`

Errors: On error, -1 is returned.

See also: fpSocket (1106), fpSend (1104), fpRecvFrom (1104)

26.4.16 fpsetsockopt

Synopsis: Set socket options.

Declaration:
```
function fpsetsockopt(s: cint;level: cint;optname: cint;optval: pointer;
                      optlen: tsocklen) : cint
```

Visibility: default

Description: `fpSetSockOpt` sets the connection options for socket `S`. 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.

XXX set `Level` to `XXX`, the protocol number of the protocol which should interprete the option.

The actual option is stored in a buffer pointed to by `optval`, with length `optlen`.

For more information on this call, refer to the unix manual page setsockopt

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: fpGetSockOpt (1102)

26.4.17 fpshutdown

Synopsis: Close one end of full duplex connection.

Declaration:
```
function fpshutdown(s: cint;how: cint) : cint
```

Visibility: default

Description: `fpShutDown` closes one end of a full duplex socket connection, described by `S`. The parameter `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: fpSocket (1106), fpConnect (1100)

26.4.18 fpsocket

Synopsis: Create new socket

Declaration:
```
function fpsocket(domain: cint;xtype: cint;protocol: cint) : cint
```

Visibility: default

Description: `fpSocket` creates a new socket in domain `Domain`, from type `xType` 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 fpBind (1099) call. If unsuccesfull, the function returns -1.

for an example, see Accept (1093).

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 (1115)

26.4.19 fpsocketpair

Synopsis: Create socket pair.

Declaration:
```
function fpsocketpair(d: cint;xtype: cint;protocol: cint;sv: pcint)
                            : cint
```

Visibility: default

Description: `fpSocketPair` creates 2 sockets in domain `D`, from type `xType` and using protocol `Protocol`. The pair is returned in `sv`, 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 (1114)

See also: Str2UnixSockAddr (1115)

26.4.20 GetPeerName

Synopsis: Return the name (address) of the connected peer (deprecated).

Declaration:
```
function GetPeerName(Sock: LongInt;var Addr;var Addrlen: LongInt)
                            : LongInt
```

Visibility: default

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 (1096), Socket (1114)

26.4.21 GetSocketName

Synopsis: Return name of socket (deprecated).

Declaration:
```
function GetSocketName(Sock: LongInt;var Addr;var Addrlen: LongInt)
                              : LongInt
```

Visibility: default

Description: `GetSocketName` 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 (1095)

26.4.22 GetSocketOptions

Synopsis: Get current socket options (deprecated).

Declaration:
```
function GetSocketOptions(Sock: LongInt;Level: LongInt;OptName: LongInt;
                                 var OptVal;var optlen: LongInt) : LongInt
```

Visibility: default

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 \seem{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: SetSocketOptions (1112)

26.4.23 HostAddrToStr

Synopsis: Convert a host address to a string.

Declaration:
```
function HostAddrToStr(Entry: in_addr) : AnsiString
```

Visibility: default

Description: `HostAddrToStr` converts the host address in `Entry` to a string representation in human-readable form (a dotted quad).

Basically, it is the same as NetAddrToStr (1109), but with the bytes in correct order.

See also: NetAddrToStr (1109), StrToHostAddr (1115), StrToNetAddr (1116)

26.4.24 HostAddrToStr6

Synopsis: Convert a IPV6 host address to a string representation.

Declaration:
```
function HostAddrToStr6(Entry: Tin6_addr) : AnsiString
```

Visibility: default

Description: `HostAddrToStr6` converts the IPV6 host address in `Entry` to a string representation in human-readable form.

Basically, it is the same as NetAddrToStr6 (1109), but with the bytes in correct order.

See also: NetAddrToStr (1109), StrToHostAddr (1115), StrToNetAddr (1116), StrToHostAddr6 (1116)

26.4.25 HostToNet

Synopsis: Convert a host address to a network address

Declaration:
```
function HostToNet(Host: in_addr) : in_addr
function HostToNet(Host: LongInt) : LongInt
```

Visibility: default

Description: `HostToNet` converts a host address to a network address. It takes care of endianness of the host machine. The address can be specified as a dotted quad or as a longint.

Errors: None.

See also: NetToHost (1110), NToHS (1110), HToNS (1109), ShortHostToNet (1113), ShortNetToHost (1113)

26.4.26 htonl

Synopsis: Convert long integer from host ordered to network ordered

Declaration:
```
function htonl(host: LongInt) : LongInt
```

Visibility: default

Description: `htonl` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: htons (1109), ntohl (1110), ntohs (1110)

26.4.27 htons

Synopsis: Convert short integer from host ordered to network ordered

Declaration: `function htons(host: Word) : Word`

Visibility: default

Description: `htons` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: htonl (1108), ntohl (1110), ntohs (1110)

26.4.28 Listen

Synopsis: Listen for connections on socket (deprecated).

Declaration: `function Listen(Sock: LongInt;MaxConnect: LongInt) : Boolean`

Visibility: default

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 (1114), Bind (1095), Connect (1096)

26.4.29 NetAddrToStr

Synopsis: Convert a network address to a string.

Declaration: `function NetAddrToStr(Entry: in_addr) : AnsiString`

Visibility: default

Description: `NetAddrToStr` converts the network address in `Entry` to a string representation in human-readable form (a dotted quad).

See also: HostAddrToStr (1108), StrToNetAddr (1116), StrToHostAddr (1115)

26.4.30 NetAddrToStr6

Synopsis: Convert a IPV6 network address to a string.

Declaration: `function NetAddrToStr6(Entry: Tin6_addr) : AnsiString`

Visibility: default

Description: `NetAddrToStr6` converts the IPV6 network address in `Entry` to a string representation in human-readable form.

Basically, it is the same as NetAddrToStr6 (1109), but with the bytes in correct order.

See also: NetAddrToStr (1109), StrToHostAddr (1115), StrToNetAddr (1116), StrToHostAddr6 (1116)

26.4.31 NetToHost

Synopsis: Convert a network address to a host address.

Declaration:
```
function NetToHost(Net: in_addr) : in_addr
function NetToHost(Net: LongInt) : LongInt
```

Visibility: default

Description: `NetToHost` converts a network address to a host address. It takes care of endianness of the host machine. The address can be specified as a dotted quad or as a longint.

Errors: None.

See also: HostToNet (1108), NToHS (1110), HToNS (1109), ShortHostToNet (1113), ShortNetToHost (1113)

26.4.32 NToHl

Synopsis: Convert long integer from network ordered to host ordered

Declaration:
```
function NToHl(Net: LongInt) : LongInt
```

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machinen, and returns the correctly ordered result.

See also: htonl (1108), htons (1109), ntohs (1110)

26.4.33 NToHs

Synopsis: Convert short integer from network ordered to host ordered

Declaration:
```
function NToHs(Net: Word) : Word
```

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machinen, and returns the correctly ordered result.

See also: htonl (1108), htons (1109), ntohl (1110)

26.4.34 Recv

Synopsis: Receive data on socket (deprecated)

Declaration:
```
function Recv(Sock: LongInt;var Buf;BufLen: LongInt;Flags: LongInt)
                  : LongInt
```

Visibility: default

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:

1Process out-of band data.

4Bypass 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 (1111)

26.4.35 RecvFrom

Synopsis: Receive data from an unconnected socket (deprecated)

Declaration:
```
function RecvFrom(Sock: LongInt;var Buf;Buflen: LongInt;Flags: LongInt;
                  var Addr;var AddrLen: LongInt) : LongInt
```

Visibility: default

Description: `RecvFrom` receives data in buffer `Buf` with maximum length `BufLen` from socket `Sock`. Receipt is controlled by options in `Flags`. `Addr` will be filled with the address from the sender, and will have length The function returns the number of bytes received, or -1 on error. `AddrLen`.

Errors: On error, -1 is returned.

See also: Socket (1114), recv (1110), Send (1111)

26.4.36 Send

Synopsis: Send data through socket (deprecated)

Declaration:
```
function Send(Sock: LongInt;const Buf;BufLen: LongInt;Flags: LongInt)
                : LongInt
```

Visibility: default

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 (1110)

26.4.37 SendTo

Synopsis: Send data through an unconnected socket to an address (deprecated).

Declaration:
```
function SendTo(Sock: LongInt;const Buf;BufLen: LongInt;Flags: LongInt;
                var Addr;AddrLen: LongInt) : LongInt
```

Visibility: default

Description: `SendTo` sends data from buffer `Buf` with length `Buflen` through socket `Sock` with options `Flags`. The data is sent to address `Addr`, which has length `AddrLen`

Errors: On error, -1 is returned.

See also: Socket (1114), Send (1111), RecvFrom (1111)

26.4.38 SetSocketOptions

Synopsis: Set socket options (deprecated).

Declaration:
```
function SetSocketOptions(Sock: LongInt;Level: LongInt;OptName: LongInt;
                          const OptVal;optlen: LongInt) : LongInt
```

Visibility: default

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

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 (1107)

26.4.39 ShortHostToNet

Synopsis: Convert a host port number to a network port number

Declaration: `function ShortHostToNet(Host: Word) : Word`

Visibility: default

Description: `ShortHostToNet` converts a host port number to a network port number. It takes care of endianness of the host machine.

Errors: None.

See also: ShortNetToHost (1113), HostToNet (1108), NToHS (1110), HToNS (1109)

26.4.40 ShortNetToHost

Synopsis: Convert a network port number to a host port number

Declaration: `function ShortNetToHost(Net: Word) : Word`

Visibility: default

Description: `ShortNetToHost` converts a network port number to a host port number. It takes care of endianness of the host machine.

Errors: None.

See also: ShortNetToHost (1113), HostToNet (1108), NToHS (1110), HToNS (1109)

26.4.41 Shutdown

Synopsis: Close one end of full duplex connection (deprecated).

Declaration: `function Shutdown(Sock: LongInt;How: LongInt) : LongInt`

Visibility: default

Description: `fpShutDown` closes one end of a full duplex socket connection, described by `Sock`. The parameter `How` determines how the connection will be shut down, and can be one of the following:

0Further receives are disallowed.

1Further sends are disallowed.

2Sending 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 (1114), Connect (1096)

26.4.42 Sock2File

Synopsis: Convert socket to untyped file descriptors

Declaration:
```
procedure Sock2File(Sock: LongInt;var SockIn: File;var SockOut: File)
```

Visibility: default

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 (1114), Sock2Text (1114)

26.4.43 Sock2Text

Synopsis: Convert socket to text file descriptors

Declaration:
```
procedure Sock2Text(Sock: LongInt;var SockIn: Text;var SockOut: Text)
```

Visibility: default

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 (1114), Sock2File (1114)

26.4.44 Socket

Synopsis: Create new socket (deprecated)

Declaration:
```
function Socket(Domain: LongInt;SocketType: LongInt;Protocol: LongInt)
                    : LongInt
```

Visibility: default

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 (1095) call. If unsuccesfull, the function returns -1.

for an example, see Accept (1093).

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 (1115)

26.4.45 socketerror

Synopsis: Contains the error code for the last socket operation.

Declaration:
```
function socketerror : cint
```

Visibility: default

Description: `SocketError` contains the error code for the last socket operation. It can be examined to return the last socket error.

26.4.46 SocketPair

Synopsis: Create socket pair (deprecated).

Declaration:
```
function SocketPair(Domain: LongInt;SocketType: LongInt;
                    Protocol: LongInt;var Pair: TSockArray) : LongInt
```

Visibility: default

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 (1114)

See also: Str2UnixSockAddr (1115)

26.4.47 Str2UnixSockAddr

Synopsis: Convert path to TUnixSockAddr (1093)

Declaration:
```
procedure Str2UnixSockAddr(const addr: String;var t: TUnixSockAddr;
                           var len: LongInt)
```

Visibility: default

Description: `Str2UnixSockAddr` transforms a Unix socket address in a string to a `TUnixSockAddr` structure which can be passed to the Bind (1095) call.

Errors: None.

See also: Socket (1114), Bind (1095)

26.4.48 StrToHostAddr

Synopsis: Convert a string to a host address.

Declaration:
```
function StrToHostAddr(IP: AnsiString) : in_addr
```

Visibility: default

Description: `StrToHostAddr` converts the string representation in `IP` to a host address and returns the host address.

Errors: On error, the host address is filled with zeroes.

See also: NetAddrToStr (1109), HostAddrToStr (1108), StrToNetAddr (1116)

26.4.49 StrToHostAddr6

Synopsis: Convert a string to a IPV6 host address.

Declaration: `function StrToHostAddr6(IP: String) : Tin6_addr`

Visibility: default

Description: `StrToHostAddr6` converts the string representation in `IP` to a IPV6 host address and returns the host address.

Errors: On error, the address is filled with zeroes.

See also: NetAddrToStr6 (1109), HostAddrToStr6 (1108), StrToHostAddr (1115)

26.4.50 StrToNetAddr

Synopsis: Convert a string to a network address.

Declaration: `function StrToNetAddr(IP: AnsiString) : in_addr`

Visibility: default

Description: `StrToNetAddr` converts the string representation in `IP` to a network address and returns the network address.

Errors: On error, the network address is filled with zeroes.

See also: NetAddrToStr (1109), HostAddrToStr (1108), StrToHostAddr (1115)

26.4.51 StrToNetAddr6

Synopsis: Convert a string to a IPV6 network address

Declaration: `function StrToNetAddr6(IP: AnsiString) : Tin6_addr`

Visibility: default

Description: `StrToNetAddr6` converts the string representation in `IP` to a IPV6 network address and returns the network address.

Errors: On error, the address is filled with zeroes.

See also: NetAddrToStr6 (1109), HostAddrToStr6 (1108), StrToHostAddr6 (1116)

Chapter 27

Reference for unit 'strings'

27.1 Overview

This chapter describes the `STRINGS` unit for Free Pascal. This unit is system independent, and therefore works on all supported platforms.

27.2 Procedures and functions

27.2.1 stralloc

Synopsis: Allocate memory for a new null-terminated string on the heap

Declaration: `function stralloc(L: SizeInt) : pchar`

Visibility: default

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

Errors: If there is not enough memory, a run-time error occurs.

See also: StrNew (1125), StrPCopy (1127)

27.2.2 strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function strcat(dest: pchar;source: pchar) : pchar`

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: StrLCat (1121)

Listing: ./stringex/ex11.pp

```
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);
  StrDispose(P2);
end.
```

27.2.3 strcomp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration: `function strcomp(str1: pchar;str2: pchar) : SizeInt`

Visibility: default

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`.

For an example, see StrLComp (1122).

Errors: None.

See also: StrLComp (1122), StrIComp (1121), StrLIComp (1124)

27.2.4 strcopy

Synopsis: Copy a null-terminated string

Declaration: `function strcopy(dest: pchar;source: pchar) : pchar`

Visibility: default

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 (1127), StrLCopy (1123), StrECopy (1119)

Listing: ./stringex/ex4.pp

```
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
    WriteIn ('Oh-oh problems...')
  else
    WriteIn ('All is well : PP=',PP);
  StrDispose(PP);
end.
```

27.2.5 strdispose

Synopsis: disposes of a null-terminated string on the heap

Declaration: `procedure strdispose(p: pchar)`

Visibility: default

Description: Removes the string in `P` from the heap and releases the memory.

Errors: None.

See also: StrNew (1125)

Listing: ./stringex/ex17.pp

```
Program Example17;

Uses strings;

{ Program to demonstrate the StrDispose function. }

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

var P2 : PChar;

begin
  P2:=StrNew (P1);
  WriteIn ('P2 : ',P2);
  StrDispose(P2);
end.
```

27.2.6 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strecopy(dest: pchar;source: pchar) : pchar`

Visibility: default

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 (1123), StrCopy (1118)

Listing: ./stringex/ex6.pp

```
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
    WriteIn('Something is wrong here !')
  else
    WriteIn ('PP= ',PP);
  StrDispose(PP);
end.
```

27.2.7 strend

Synopsis: Return a pointer to the end of a null-terminated string

Declaration: `function strend(p: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the end of `P`. (i.e. to the terminating null-character.

Errors: None.

See also: StrLen (1123)

Listing: ./stringex/ex7.pp

```
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
    WriteIn('Something is wrong here !')
```

```
  else
    WriteIn ('All is well..');
end.
```

27.2.8 stricomp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration: `function stricomp(str1: pchar;str2: pchar) : SizeInt`

Visibility: default

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 (1122), StrComp (1118), StrLIComp (1124)

Listing: ./stringex/ex8.pp

```
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);
  WriteIn (l,' characters are the same.');
end.
```

27.2.9 strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function strlcat(dest: pchar;source: pchar;l: SizeInt) : pchar`

Visibility: default

Description: Adds `MaxLen` characters from `Source` to `Dest`, and adds a terminating null-character. Returns `Dest`.

Errors: None.

See also: StrCat (1117)

Listing: ./stringex/ex12.pp

```
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);
  WriteIn ('P2 = ',P2);
  StrDispose(P2)
end.
```

27.2.10 strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings

Declaration: `function strlcomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

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 (1118), StrIComp (1121), StrLIComp (1124)

Listing: ./stringex/ex8.pp

```
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.
```

27.2.11 strlcopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: `function strlcopy(dest: pchar;source: pchar;maxlen: SizeInt) : pchar`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`, and makes `Dest` a null terminated string.

Errors: No length checking is performed.

See also: StrCopy (1118), StrECopy (1119)

Listing: ./stringex/ex5.pp

```
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));
  StrDispose(PP);
end.
```

27.2.12 strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: pchar) : sizeint`

Visibility: default

Description: Returns the length of the null-terminated string `P`.

Errors: None.

See also: StrNew (1125)

Listing: ./stringex/ex1.pp

```
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.
```

27.2.13 strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration: `function strlicomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

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`.

For an example, see StrIComp (1121)

Errors: None.

See also: StrLComp (1122), StrComp (1118), StrIComp (1121)

27.2.14 strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration: `function strlower(p: pchar) : pchar`

Visibility: default

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

Errors: None.

See also: StrUpper (1129)

Listing: ./stringex/ex14.pp

```
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.
```

27.2.15 strmove

Synopsis: Move a null-terminated string to new location.

Declaration: `function strmove(dest: pchar;source: pchar;l: SizeInt) : pchar`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: StrLCopy (1123), StrCopy (1118)

Listing: ./stringex/ex10.pp

```
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);
  StrDispose(P2);
end.
```

27.2.16 strnew

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: pchar) : pchar`

Visibility: default

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: StrCopy (1118), StrDispose (1119)

Listing: ./stringex/ex16.pp

```
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);
  StrDispose(P2);
end.
```

27.2.17 strpas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function strpas(p: pchar) : shortstring`

Visibility: default

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 (1127)

Listing: ./stringex/ex3.pp

```
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.
```

27.2.18 strpcopy

Synopsis: Copy a pascal string to a null-terminated string

Declaration: `function strpcopy(d: pchar;const s: String) : pchar`

Visibility: default

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 (1126)

Listing: ./stringex/ex2.pp

```
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
    WriteIn ('This is impossible !!')
  else
    writeln (P);
  StrDispose(P);
end.
```

27.2.19 strpos

Synopsis: Search for a null-terminated substring in a null-terminated string

Declaration: `function strpos(str1: pchar;str2: pchar) : pchar`

Visibility: default

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: StrScan (1128), StrRScan (1128)

Listing: ./stringex/ex15.pp

```
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.
```

27.2.20 strrscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function strrscan(p: pchar;c: Char) : pchar`

Visibility: default

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`.

For an example, see StrScan (1128).

Errors: None.

See also: StrScan (1128), StrPos (1127)

27.2.21 strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: pchar;c: Char) : pchar`

Visibility: default

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: StrRScan (1128), StrPos (1127)

Listing: ./stringex/ex13.pp

```
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.
```

27.2.22 strupper

Synopsis: Convert null-terminated string to all-uppercase

Declaration: `function strupper(p: pchar) : pchar`

Visibility: default

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

For an example, see StrLower (1124)

Errors: None.

See also: StrLower (1124)

Chapter 28

Reference for unit 'strutils'

28.1 Used units

Table 28.1: Used units by unit 'strutils'

Name	Page
SysUtils	1130

28.2 Constants, types and variables

28.2.1 Resource strings

```
SErrAmountStrings = 'Amount of search and replace strings don''t match'
```

Error message used in stringsreplace function

28.2.2 Constants

```
AnsiResemblesProc : TCompareTextProc = @SoundexProc
```

This procedural variable is standard set to SoundexProc (1154) but can be overriden with a user-defined algorithm. This algorithm should return `True` if `AText` resembles `AOtherText`, or `False` otherwise. The standard routine compares the soundexes of the two strings and returns `True` if they are equal.

```
Brackets = ['(',')','[',']','{','}']
```

Set of characters that contain all possible bracket characters

```
DigitChars = ['0'..'9']
```

Set of digit characters

```
StdSwitchChars = ['-','/']
```

Standard characters for the SwitchChars argument of GetCmdLineArg (1143).

```
StdWordDelims = [#0..' ',',',',','.',';','/','\',':','''',',"',',`'] + Brackets
```

Standard word delimiter values.

```
WordDelimiters : Set of Char = [#0..#255] - ['a'..'z','A'..'Z','1'..'9','0']
```

Standard word delimiters, used in the SearchBuf (1153) call.

28.2.3 Types

```
TCompareTextProc = function(const AText: String;const AOther: String)
                            : Boolean
```

Function prototype for comparing two string in AnsiResemblesText (1136)

```
TSoundexIntLength = 1..8
```

Range of allowed integer soundex lengths.

```
TSoundexLength = 1..MaxInt
```

Range of allowed soundex lengths.

```
TStringSeachOption = TStringSearchOption
```

There is an typo error in the original Borland StrUtils unit. This type just refers to the correct TStringSearchOption (1131) and is provided for compatibility only.

```
TStringSearchOption = (soDown,soMatchCase,soWholeWord)
```

Table 28.2: Enumeration values for type TStringSearchOption

Value	Explanation
soDown	Search in down direction.
soMatchCase	Match case
soWholeWord	Search whole words only.

Possible options for SearchBuf (1153) call.

```
TStringSearchOptions= Set of (soDown,soMatchCase,soWholeWord)
```

Set of options for SearchBuf (1153) call.

28.3 Procedures and functions

28.3.1 AddChar

Synopsis: Add characters to the left of a string till a certain length

Declaration:
```
function AddChar(C: Char;const S: String;N: Integer) : String
```

Visibility: default

Description: `AddChar` adds characters (`C`) to the left of S till the length `N` is reached, and returns the resulting string. If the length of `S` is already equal to or larger than `N` , then no characters are added. The resulting string can be thought of as a right-aligned version of `S` , with length `N` .

Errors: None

See also: AddCharR (1132), PadLeft (1148), PadRight (1149), PadCenter (1148)

28.3.2 AddCharR

Synopsis: Add chars at the end of a string till it reaches a certain length

Declaration:
```
function AddCharR(C: Char;const S: String;N: Integer) : String
```

Visibility: default

Description: `AddCharR` adds characters (`C`) to the right of S till the length `N` is reached, and returns the resulting string. If the length of `S` is already equal to or larger than `N`, then no characters are added. The resulting string can be thought of as a left-aligned version of `S`, with length `N` .

Errors: None

See also: AddChar (1132)

28.3.3 AnsiContainsStr

Synopsis: Checks whether a string contains a given substring

Declaration:
```
function AnsiContainsStr(const AText: String;const ASubText: String)
                                 : Boolean
```

Visibility: default

Description: `AnsiContainsString` checks whether `AText` contains `ASubText`, and returns `True` if this is the case, or returns `False` otherwise. The search is performed case-sensitive.

Errors: None

See also: AnsiContainsText (1132), AnsiEndsStr (1133), AnsiIndexStr (1133), AnsiStartsStr (1137)

28.3.4 AnsiContainsText

Synopsis: Check whether a string contains a certain substring, ignoring case.

Declaration:
```
function AnsiContainsText(const AText: String;const ASubText: String)
                                 : Boolean
```

Visibility: default

Description: `AnsiContainsString` checks whether `AText` contains `ASubText`, and returns `True` if this is the case, or returns `False` otherwise. The search is performed case-insensitive.

Errors:

See also: AnsiContainsStr (1132), AnsiEndsText (1133), AnsiIndexText (1134), AnsiStartsText (1137)

28.3.5 AnsiEndsStr

Synopsis: Check whether a string ends with a certain substring

Declaration:
```
function AnsiEndsStr(const ASubText: String;const AText: String)
                            : Boolean
```

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it ends with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None.

See also: AnsiEndsText (1133), AnsiStartsStr (1137), AnsiIndexStr (1133), AnsiContainsStr (1132)

28.3.6 AnsiEndsText

Synopsis: Check whether a string ends with a certain substring, ignoring case.

Declaration:
```
function AnsiEndsText(const ASubText: String;const AText: String)
                             : Boolean
```

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it ends with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None

See also: AnsiStartsText (1137), AnsiEndsStr (1133), AnsiIndexText (1134), AnsiContainsText (1132)

28.3.7 AnsiIndexStr

Synopsis: Searches, observing case, for a string in an array of strings.

Declaration:
```
function AnsiIndexStr(const AText: String;
                            const AValues: Array of String) : Integer
```

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues` . If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched observing case.

Errors: None.

See also: AnsiIndexText (1134), AnsiMatchStr (1134), AnsiMatchText (1135)

28.3.8 AnsiIndexText

Synopsis: Searches, case insensitive, for a string in an array of strings.

Declaration:
```
function AnsiIndexText(const AText: String;
                       const AValues: Array of String) : Integer
```

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues` . If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched ignoring case.

Errors: None

See also: AnsiIndexStr (1133), AnsiMatchStr (1134), AnsiMatchText (1135)

28.3.9 AnsiLeftStr

Synopsis: Copies a number of characters starting at the left of a string

Declaration:
```
function AnsiLeftStr(const AText: AnsiString;const ACount: Integer)
                     : AnsiString
```

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` leftmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `LeftStr` function.

Errors: None.

See also: AnsiMidStr (1135), AnsiRightStr (1137), LeftStr (1146), RightStr (1151), MidStr (1147), LeftBStr (1146), RightBStr (1151), MidBStr (1147)

28.3.10 AnsiMatchStr

Synopsis: Check whether a string occurs in an array of strings, observing case.

Declaration:
```
function AnsiMatchStr(const AText: String;
                      const AValues: Array of String) : Boolean
```

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched observing case.

This function simply calls AnsiIndexStr (1133) and checks whether it returns -1 or not.

Errors:

28.3.11 AnsiMatchText

Synopsis: Check whether a string occurs in an array of strings, disregarding case.

Declaration:
```
function AnsiMatchText(const AText: String;
                       const AValues: Array of String) : Boolean
```

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched ignoring case.

This function simply calls AnsiIndexText (1134) and checks whether it returns -1 or not.

Errors:

28.3.12 AnsiMidStr

Synopsis: Returns a number of characters copied from a given location in a string

Declaration:
```
function AnsiMidStr(const AText: AnsiString;const AStart: Integer;
                    const ACount: Integer) : AnsiString
```

Visibility: default

Description: `AnsiMidStr` returns `ACount` characters from `AText`, starting at position `AStart`. If `AStart+ACount` is larger than the length of `AText`, only as much characters as available in `AText` (starting from `AStart`) will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `MidStr` function.

Errors: None

See also: AnsiLeftStr (1134), AnsiRightStr (1137), LeftStr (1146), RightStr (1151), MidStr (1147), LeftBStr (1146), RightBStr (1151), MidBStr (1147)

28.3.13 AnsiProperCase

Synopsis: Pretty-Print a string: make lowercase and capitalize first letters of words

Declaration:
```
function AnsiProperCase(const S: String;const WordDelims: TSysCharSet)
                        : String
```

Visibility: default

Description: `AnsiProperCase` converts `S` to an all lowercase string, but capitalizes the first letter of every word in the string, and returns the resulting string. When searching for words, the characters in `WordDelimiters` are used to determine the boundaries of words. The constant StdWordDelims (1131) can be used for this.

Errors:

28.3.14 AnsiReplaceStr

Synopsis: Search and replace all occurrences of a string, case sensitive.

Declaration:
```
function AnsiReplaceStr(const AText: String;const AFromText: String;
                        const AToText: String) : String
```

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText`, and returns the resulting string. The search is performed observing case.

Errors: None.

See also: AnsiReplaceText (1136), SearchBuf (1153)

28.3.15 AnsiReplaceText

Synopsis: Search and replace all occurrences of a string, case insensitive.

Declaration:
```
function AnsiReplaceText(const AText: String;const AFromText: String;
                         const AToText: String) : String
```

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText` , and returns the resulting string. The search is performed ignoring case.

Errors: None.

See also: AnsiReplaceStr (1136), SearchBuf (1153)

28.3.16 AnsiResemblesText

Synopsis: Check whether 2 strings resemble each other.

Declaration:
```
function AnsiResemblesText(const AText: String;const AOther: String)
                           : Boolean
```

Visibility: default

Description: `AnsiResemblesText` will check whether AnsiResemblesProc (1130) is set. If it is not set, `False` is returned. If it is set, `AText` and `AOtherText` are passed to it and it's result is returned.

Errors: None.

See also: AnsiResemblesProc (1130), SoundexProc (1154)

28.3.17 AnsiReverseString

Synopsis: Reverse the letters in a string.

Declaration:
```
function AnsiReverseString(const AText: AnsiString) : AnsiString
```

Visibility: default

Description: `AnsiReverseString` returns a string with all characters of `AText` in reverse order.

if the result of this function equals `AText`, `AText` is called an anagram.

Errors: None.

28.3.18 AnsiRightStr

Synopsis: Copies a number of characters starting at the right of a string

Declaration:
```
function AnsiRightStr(const AText: AnsiString;const ACount: Integer)
                      : AnsiString
```

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` rightmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `RightStr` function.

Errors: None.

See also: AnsiLeftStr (1134), AnsiMidStr (1135), LeftStr (1146), RightStr (1151), MidStr (1147), LeftBStr (1146), RightBStr (1151), MidBStr (1147)

28.3.19 AnsiStartsStr

Synopsis: Check whether a string starts with a given substring, observing case

Declaration:
```
function AnsiStartsStr(const ASubText: String;const AText: String)
                       : Boolean
```

Visibility: default

Description: `AnsiStartsStr` checks `AText` to see whether it starts with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals 1.

Errors:

See also: AnsiEndsStr (1133), AnsiStartsStr (1137), AnsiIndexStr (1133), AnsiContainsStr (1132)

28.3.20 AnsiStartsText

Synopsis: Check whether a string starts with a given substring, ignoring case

Declaration:
```
function AnsiStartsText(const ASubText: String;const AText: String)
                        : Boolean
```

Visibility: default

Description: `AnsiStartsText` checks `AText` to see whether it starts with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals 1.

Errors: None.

See also: AnsiEndsText (1133), AnsiStartsStr (1137), AnsiIndexText (1134), AnsiContainsText (1132)

28.3.21 BinToHex

Synopsis: Convert a binary buffer to a hexadecimal string

Declaration: `procedure BinToHex(BinValue: PChar;HexValue: PChar;BinBufSize: Integer)`

Visibility: default

Description: `BinToHex` converts the byte values in `BinValue` to a string consisting of 2-charachter hexadecimal strings in `HexValue`. `BufSize` specifies the length of `BinValue`, which means that `HexValue` must have size `2*BufSize`.

For example a buffer containing the byte values 255 and 0 will be converted to FF00.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: HexToBin (1144)

28.3.22 Copy2Space

Synopsis: Returns all characters in a string till the first space character (not included).

Declaration: `function Copy2Space(const S: String) : String`

Visibility: default

Description: `Copy2Space` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. The string `S` is left untouched. If there is no space in `S`, then the whole string `S` is returned.

This function simply calls Copy2Symb (1139) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: Copy2Symb (1139), Copy2SpaceDel (1138)

28.3.23 Copy2SpaceDel

Synopsis: Deletes and returns all characters in a string till the first space character (not included).

Declaration: `function Copy2SpaceDel(var S: String) : String`

Visibility: default

Description: `Copy2SpaceDel` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. All returned characters, including the space, are deleted from the string `S`, after which it is right-trimmed. If there is no space in `S`, then the whole string `S` is returned, and `S` itself is emptied.

This function simply calls Copy2SymbDel (1139) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: Copy2SymbDel (1139), Copy2Space (1138)

28.3.24 Copy2Symb

Synopsis: Returns all characters in a string till a given character (not included).

Declaration: `function Copy2Symb(const S: String;Symb: Char) : String`

Visibility: default

Description: `Copy2SymbDel` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. The string `S` is left untouched. If `Symb` does not appear in `S`, then the whole of `S` is returned.

Errors: None.

See also: Copy2Space (1138), Copy2SymbDel (1139)

28.3.25 Copy2SymbDel

Synopsis: Deletes and returns all characters in a string till a given character (not included).

Declaration: `function Copy2SymbDel(var S: String;Symb: Char) : String`

Visibility: default

Description: `Copy2SymbDel` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. All returned characters, *not* including the `Symb` character, are deleted from the string `S`, after which it is right-trimmed. If `Symb` does not appear in `S`, then the whole of `S` is returned, and `S` itself is emptied.

Errors: None.

See also: Copy2SpaceDel (1138), Copy2Symb (1139)

28.3.26 Dec2Numb

Synopsis: Convert a decimal number to a string representation, using given a base.

Declaration: `function Dec2Numb(N: LongInt;Len: Byte;Base: Byte) : String`

Visibility: default

Description: `Dec2Numb` converts `N` to its representation using base `Base` . The resulting string is left-padded with zeroes till it has length `Len` . `Base` must be in the range 2-36 to be meaningful, but no checking on this is performed.

Errors: If `Base` is out of range, the resulting string will contain unreadable (non-alphanumeric) characters.

See also: Hex2Dec (1143), IntToBin (1144), intToRoman (1145), RomanToInt (1152)

28.3.27 DecodeSoundexInt

Synopsis: Decodes the integer representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexInt(AValue: Integer) : String`

Visibility: default

Description: `DecodeSoundexInt` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the SoundexInt (1154) function. The result is the soundex string corresponding to `AValue`. .

Errors: None.

See also: SoundexInt (1154), DecodeSoundexWord (1140), Soundex (1153)

28.3.28 DecodeSoundexWord

Synopsis: Decodes the word-sized representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexWord(AValue: Word) : String`

Visibility: default

Description: `DecodeSoundexWord` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the SoundexWord (1155) function. The result is the soundex string corresponding to `AValue` .

Errors: None.

See also: SoundexInt (1154), DecodeSoundexInt (1139), Soundex (1153)

28.3.29 DelChars

Synopsis: Delete all occurrences of a given character from a string.

Declaration: `function DelChars(const S: String;Chr: Char) : String`

Visibility: default

Description: `DelChars` returns a copy of `S` with all `Chr` characters removed from it.

Errors: None.

See also: DelSpace (1140), DelSpace1 (1140)

28.3.30 DelSpace

Synopsis: Delete all occurrences of a space from a string.

Declaration: `function DelSpace(const S: String) : String`

Visibility: default

Description: `DelSpace` returns a copy of `S` with all spaces (ASCII code 32) removed from it.

Errors: None.

See also: DelChars (1140), DelSpace1 (1140)

28.3.31 DelSpace1

Synopsis: Reduces sequences of space characters to 1 space character.

Declaration: `function DelSpace1(const S: String) : String`

Visibility: default

Description: `DelSpace1` returns a copy of `S` with all sequences of spaces reduced to 1 space.

Errors: None.

See also: DelChars (1140), DelSpace (1140)

28.3.32 DupeString

Synopsis: Creates and concatenates N copies of a string

Declaration:
```
function DupeString(const AText: String;ACount: Integer) : String
```

Visibility: default

Description: `DupeString` returns a string consisting of `ACount` concatenations of `AText`. Thus

```
DupeString('1234567890',3);
```

will produce a string

```
'123456789012345678901234567890'
```

Errors: None.

28.3.33 ExtractDelimited

Synopsis: Extract the N-th delimited part from a string.

Declaration:
```
function ExtractDelimited(N: Integer;const S: String;
                          const Delims: TSysCharSet) : String
```

Visibility: default

Description: `ExtractDelimited` extracts the `N`-th part from the string `S`. The set of characters in `Delims` are used to mark part boundaries. When a delimiter is encountered, a new part is started and the old part is ended. Another way of stating this is that any (possibly empty) series of characters not in `Delims`, situated between 2 characters in `Delims`, it is considered as piece of a part. This means that if 2 delimiter characters appear next to each other, there is an empty part between it. If an `N`-th part cannot be found, an empty string is returned. However, unlike ExtractWord (1142), an empty string is a valid return value, i.e. a part can be empty.

The pre-defined constant StdWordDelims (1131) can be used for the `Delims` argument. The pre-defined constant Brackets (1130) would be better suited the `Delims` argument e.g. in case factors in a mathematical expression are searched.

Errors: None.

See also: ExtractSubStr (1141), ExtractWord (1142), ExtractWordPos (1142)

28.3.34 ExtractSubstr

Synopsis: Extract a word from a string, starting at a given position in the string.

Declaration:
```
function ExtractSubstr(const S: String;var Pos: Integer;
                       const Delims: TSysCharSet) : String
```

Visibility: default

Description: `ExtractSubStr` returns all characters from `S` starting at position `Pos` till the first character in `Delims`, or till the end of `S` is reached. The delimiter character is not included in the result. `Pos` is then updated to point to the next first non-delimiter character in `S`. If `Pos` is larger than the Length of `S`, an empty string is returned.

The pre-defined constant StdWordDelims (1131) can be used for the `Delims` argument.

Errors: None.

See also: ExtractDelimited (1141), ExtractWord (1142), ExtractWordPos (1142)

28.3.35 ExtractWord

Synopsis: Extract the N-th word out of a string.

Declaration:
```
function ExtractWord(N: Integer;const S: String;
                     const WordDelims: TSysCharSet) : String
```

Visibility: default

Description: `ExtractWord` extracts the `N`-th word from the string `S`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims`: if a character is not in `WordDelims`, it is considered as part of a word. If an `N`-th word cannot be found, an empty string is returned.

Unlike ExtractDelimited (1141), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The pre-defined constant StdWordDelims (1131) can be used for the `WordDelims` argument.

Errors: None.

See also: ExtractWordPos (1142), ExtractSubStr (1141), ExtractDelimited (1141), IsWordPresent (1146), WordCount (1157), WordPosition (1157)

28.3.36 ExtractWordPos

Synopsis: Extract a word from a string, and return the position where it was located in the string.

Declaration:
```
function ExtractWordPos(N: Integer;const S: String;
                        const WordDelims: TSysCharSet;var Pos: Integer)
                         : String
```

Visibility: default

Description: `ExtractWordPos` extracts the `N`-th word from the string `S` and returns the position of this word in `Pos`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims` : if a character is not in `WordDelims` , it is considered as part of a word. If an `N` -th word cannot be found, an empty string is returned and `Pos` is zero.

Unlike ExtractDelimited (1141), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The pre-defined constant StdWordDelims (1131) can be used for the `WordDelims` argument.

Errors: None.

See also: ExtractWord (1142), ExtractSubStr (1141), IsWordPresent (1146), WordCount (1157), WordPosition (1157)

28.3.37 FindPart

Synopsis: Search for a substring in a string, using wildcards.

Declaration:
```
function FindPart(const HelpWilds: String;const InputStr: String)
                  : Integer
```

Visibility: default

Description: `FindPart` searches the string `InputStr` and returns the first string that matches the wildcards specification in `HelpWilds` . If no match is found, and empty string is returned. Currently, the only valid wildcards is the "?" character.

Errors: None.

See also: SearchBuf (1153)

28.3.38 GetCmdLineArg

Synopsis: Returns the command-line argument following the given switch.

Declaration:
```
function GetCmdLineArg(const Switch: String;SwitchChars: TSysCharSet)
                              : String
```

Visibility: default

Description: `GetCmdLineArg` returns the value for the `Switch` option on the command-line, if any is given. Command-line arguments are considered switches if they start with one of the characters in the `SwitchChars` set. The value is the command-line argument following the switch command-line argument.

Gnu-style (long) Options of the form switch=value are not supported.

The StdSwitchChars (1131) constant can be used as value for the `SwitchChars` parameter.

Errors: The GetCmdLineArg does not check whether the value of the option does not start with a switch character. i.e.

```
myprogram -option1 -option2
```

will result in "-option2" as the result of the `GetCmdLineArg` call for option1.

See also: StdSwitchChars (1131)

28.3.39 Hex2Dec

Synopsis: Converts a hexadecimal string to a decimal value

Declaration:
```
function Hex2Dec(const S: String) : LongInt
```

Visibility: default

Description: `Hex2Dec` converts the hexadecimal value in the string `S` to its decimal value. Unlike the standard `Val`or `StrToInt` functions, there need not be a $ sign in front of the hexadecimal value to indicate that it is indeed a hexadecimal value.

Errors: If `S` does not contain a valid hexadecimal value, an `EConvertError` exception will be raised.

See also: Dec2Numb (1139), IntToBin (1144), intToRoman (1145), RomanToInt (1152)

28.3.40 HexToBin

Synopsis: Convert a hexadecimal string to a binary buffer

Declaration:
```
function HexToBin(HexValue: PChar;BinValue: PChar;BinBufSize: Integer)
                        : Integer
```

Visibility: default

Description: `HexToBin` scans the hexadecimal string representation in `HexValue` and transforms every 2 character hexadecimal number to a byte and stores it in `BinValue`. The buffer size is the size of the binary buffer. Scanning will stop if the size of the binary buffer is reached or when an invalid character is encountered. The return value is the number of stored bytes.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: BinToHex (1138)

28.3.41 IfThen

Synopsis: Returns one of two strings, depending on a boolean expression

Declaration:
```
function IfThen(AValue: Boolean;const ATrue: String;AFalse: String)
                      : String
function IfThen(AValue: Boolean;const ATrue: String) : String
```

Visibility: default

Description: `IfThen` returns `ATrue` if `AValue` is `True`, and returns `AFalse` if `AValue` is false.

Errors: None.

See also: AnsiMatchStr (1134), AnsiMatchText (1135)

28.3.42 IntToBin

Synopsis: Converts an integer to a binary string representation, inserting spaces at fixed locations.

Declaration:
```
function IntToBin(Value: LongInt;Digits: Integer;Spaces: Integer)
                      : String
function IntToBin(Value: LongInt;Digits: Integer) : String
function intToBin(Value: Int64;Digits: Integer) : String
```

Visibility: default

Description: `IntToBin` converts `Value` to a string with it's binary (base 2) representation. The resulting string contains at least `Digits` digits, with spaces inserted every `Spaces` digits. `Spaces` should be a nonzero value. If `Digits` is larger than 32, it is truncated to 32.

Errors: If spaces is zero, a division by zero error will occur.

See also: Hex2Dec (1143), IntToRoman (1145)

28.3.43 IntToRoman

Synopsis: Represent an integer with roman numerals

Declaration:
```
function IntToRoman(Value: LongInt) : String
```

Visibility: default

Description: `IntToRoman` converts `Value` to a string with the Roman representation of `Value`. Number up to 1 million can be represented this way.

Errors: None.

See also: RomanToInt (1152), Hex2Dec (1143), IntToBin (1144)

28.3.44 IsEmptyStr

Synopsis: Check whether a string is empty, disregaring whitespace characters

Declaration:
```
function IsEmptyStr(const S: String;const EmptyChars: TSysCharSet)
                          : Boolean
```

Visibility: default

Description: `IsEmptyStr` returns `True` if the string `S` only contains characters whitespace characters, all characters in `EmptyChars` are considered whitespace characters. If a character not present in `EmptyChars` is found in `S`, `False` is returned.

Errors: None.

See also: IsWild (1145), FindPart (1142), IsWordPresent (1146)

28.3.45 IsWild

Synopsis: Check whether a string matches a wildcard search expression.

Declaration:
```
function IsWild(InputStr: String;Wilds: String;IgnoreCase: Boolean)
                     : Boolean
```

Visibility: default

Description: `IsWild` checks `InputStr` for the presence of the `Wilds` string. `Wilds` may contain "?" and "*" wildcard characters, which have their usual meaning: "*" matches any series of characters, possibly empty. "?" matches any single character. The function returns `True` if a string is found that matches `Wilds`, `False` otherwise.

If `IgnoreCase` is `True`, the non-wildcard characters are matched case insensitively. If it is `False`, case is observed when searching.

Errors: None.

See also: SearchBuf (1153), FindPart (1142)

28.3.46 IsWordPresent

Synopsis: Check for the presence of a word in a string.

Declaration:
```
function IsWordPresent(const W: String;const S: String;
                       const WordDelims: TSysCharSet) : Boolean
```

Visibility: default

Description: `IsWordPresent` checks for the presence of the word `W` in the string `S`. Words are delimited by the characters found in `WordDelims`. The function returns `True` if a match is found, `False` otherwise. The search is performed case sensitive.

This function is equivalent to the SearchBuf (1153) function with the `soWholeWords` option specified.

Errors: None.

See also: SearchBuf (1153)

28.3.47 LeftBStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration:
```
function LeftBStr(const AText: AnsiString;const AByteCount: Integer)
                  : AnsiString
```

Visibility: default

Description: `LeftBStr` returns a string containing the leftmost `AByteCount` bytes from the string `AText` . If `AByteCount` is larger than the length (in bytes) of `AText` , only as many bytes as available are returned.

Errors: None.

See also: LeftStr (1146), AnsiLeftStr (1134), RightBStr (1151), MidBStr (1147)

28.3.48 LeftStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration:
```
function LeftStr(const AText: AnsiString;const ACount: Integer)
                 : AnsiString
function LeftStr(const AText: WideString;const ACount: Integer)
                 : WideString
```

Visibility: default

Description: `LeftStr` returns a string containing the leftmost `ACount` characters from the string `AText` . If `ACount` is larger than the length (in characters) of `AText` , only as many characters as available are returned.

Errors: None.

See also: LeftBStr (1146), AnsiLeftStr (1134), RightStr (1151), MidStr (1147)

28.3.49 MidBStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration:
```
function MidBStr(const AText: AnsiString;const AByteStart: Integer;
                 const AByteCount: Integer) : AnsiString
```

Visibility: default

Description: `MidBStr` returns a string containing the first `AByteCount` bytes from the string `AText` starting at position `AByteStart`. If `AByteStart+AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned. If `AByteStart` is less than 1 or larger than the length of `AText`, then no characters are returned.

Errors: None.

See also: LeftBStr (1146), AnsiMidStr (1135), RightBStr (1151), MidStr (1147)

28.3.50 MidStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration:
```
function MidStr(const AText: AnsiString;const AStart: Integer;
                const ACount: Integer) : AnsiString
function MidStr(const AText: WideString;const AStart: Integer;
                const ACount: Integer) : WideString
```

Visibility: default

Description: `MidStr` returns a string containing the first `ACount` bytes from the string `AText` starting at position `AStart`. If `AStart+ACount` is larger than the length (in characters) of `AText`, only as many characters as available are returned. If `AStart` is less than 1 or larger than the length of `AText`, then no characters are returned.

This function is equivalent to the standard `Copy` function, and is provided for completeness only.

Errors: None.

See also: LeftStr (1146), AnsiMidStr (1135), RightStr (1151), MidBStr (1147)

28.3.51 NPos

Synopsis: Returns the position of the N-th occurence of a substring in a string.

Declaration:
```
function NPos(const C: String;S: String;N: Integer) : Integer
```

Visibility: default

Description: `NPos` checks `S` for the position of the `N`-th occurrence of `C`. If `C` occurs less than `N` times in `S`, or does not occur in `S` at all, 0 is returned. If `N` is less than 1, zero is returned.

Errors: None.

See also: WordPosition (1157), FindPart (1142)

28.3.52 Numb2Dec

Synopsis: Converts a string representation of a number to its numerical value, given a certain base.

Declaration: `function Numb2Dec(S: String;Base: Byte) : LongInt`

Visibility: default

Description: `Numb2Dec` converts the number in string `S` to a decimal value. It assumes the number is represented using `Base` as the base. No checking is performed to see whether `S` contains a valid number using base `Base`.

Errors: None.

See also: Hex2Dec (1143), Numb2USA (1148)

28.3.53 Numb2USA

Synopsis: Insert thousand separators.

Declaration: `function Numb2USA(const S: String) : String`

Visibility: default

Description: `Numb2USA` inserts thousand separators in the string `S` at the places where they are supposed to be, i.e. every 3 digits. The string `S` should contain a valid integer number, i.e. no digital number. No checking on this is done.

Errors: None.

28.3.54 PadCenter

Synopsis: Pad the string to a certain length, so the string is centered.

Declaration: `function PadCenter(const S: String;Len: Integer) : String`

Visibility: default

Description: `PadCenter` add spaces to the left and right of the string `S` till the result reaches length `Len`. If the number of spaces to add is odd, then the extra space will be added at the end. If the string `S` has length equal to or largert than `Len`, no spaces are added, and the string `S` is returned as-is.

Errors: None.

See also: PadLeft (1148), PadRight (1149), AddChar (1132), AddCharR (1132)

28.3.55 PadLeft

Synopsis: Add spaces to the left of a string till a certain length is reached.

Declaration: `function PadLeft(const S: String;N: Integer) : String`

Visibility: default

Description: `PadLeft` add spaces to the left of the string `S` till the result reaches length `Len` . If the string `S` has length equal to or largert than `Len` , no spaces are added, and the string `S` is returned as-is. The resulting string is `S` , right-justified on length `Len` .

Errors: None.

See also: PadLeft (1148), PadCenter (1148), AddChar (1132), AddCharR (1132)

28.3.56 PadRight

Synopsis: Add spaces to the right of a string till a certain length is reached.

Declaration:
```
function PadRight(const S: String;N: Integer) : String
```

Visibility: default

Description: `PadRight` add spaces to the left of the string `S` till the result reaches length `Len`. If the string `S` has length equal to or largert than `Len`, no spaces are added, and the string `S` is returned as-is. The resulting string is `S`, left-justified on length `Len` .

Errors: None.

See also: PadLeft (1148), PadCenter (1148), AddChar (1132), AddCharR (1132)

28.3.57 PosEx

Synopsis: Search for the occurance of a character in a string, starting at a certain position.

Declaration:
```
function PosEx(const SubStr: String;const S: String;Offset: Cardinal)
                  : Integer
function PosEx(const SubStr: String;const S: String) : Integer
function PosEx(c: Char;const S: String;Offset: Cardinal) : Integer
```

Visibility: default

Description: `PosEx` returns the position of the first occurrence of the character `C` or the substring `SubStr` in the string `S`, starting the search at position `Offset` (default 1). If `C` or `SubStr` does not occur in `S` after the given `Offset`, zero is returned. The position Offset is also searched.

Errors: None.

See also: NPos (1147), AnsiContainsText (1132), AnsiContainsStr (1132)

28.3.58 PosSet

Synopsis: Return the position in a string of any character out of a set of characters

Declaration:
```
function PosSet(const c: TSysCharSet;const s: ansistring) : Integer
function PosSet(const c: String;const s: ansistring) : Integer
```

Visibility: default

Description: `PosSet` returns the position in `s` of the first found character which is in the set `c`. If none of the characters in `c` is found in `s`, then `0` is returned.

Errors: None.

See also: PosEx (1149), PosSetEx (1149), #rtl.system.pos (1295), RPosEx (1152)

28.3.59 PosSetEx

Synopsis: Return the position in a string of any character out of a set of characters, starting at a certain position

Declaration:
```
function PosSetEx(const c: TSysCharSet;const s: ansistring;
                  count: Integer) : Integer
function PosSetEx(const c: String;const s: ansistring;count: Integer)
                   : Integer
```

Visibility: default

Description: `PosSetEx` returns the position in `s` of the first found character which is in the set `c`, and starts searching at character position `Count`. If none of the characters in `c` is found in `s`, then `0` is returned.

Errors: None.

See also: PosEx (1149), PosSet (1149), #rtl.system.pos (1295), RPosEx (1152)

28.3.60 RandomFrom

Synopsis: Choose a random string from an array of strings.

Declaration: `function RandomFrom(const AValues: Array of String) : String;  Overload`

Visibility: default

Description: `RandomFrom` picks at random a valid index in the array `AValues` and returns the string at that position in the array.

Errors: None.

See also: AnsiMatchStr (1134), AnsiMatchText (1135)

28.3.61 Removeleadingchars

Synopsis: Remove any leading characters in a set from a string

Declaration: `procedure Removeleadingchars(var S: AnsiString;const CSet: TSysCharset)`

Visibility: default

Description: `Removeleadingchars` removes any starting characters from `S` that appear in the set `CSet`. It stops removing characters as soon as a character not in `CSet` is encountered. This is similar in behaviour to TrimLeft (1482) which used whitespace as the set.

Errors: None.

See also: rtl.sysutils.TrimLeft (1130), RemoveTrailingChars (1151), RemovePadChars (1150), TrimLeftSet (1156)

28.3.62 RemovePadChars

Synopsis: Remove any trailing or leading characters in a set from a string

Declaration: `procedure RemovePadChars(var S: AnsiString;const CSet: TSysCharset)`

Visibility: default

Description: `RemovePadChars` removes any leading trailing characters from `S` that appear in the set `CSet`, i.e. it starts with the last character and works its way to the start of the string, and it stops removing characters as soon as a character not in `CSet` is encountered. Then the same procedure is repeated starting from the beginning of the string. This is similar in behaviour to Trim (1481) which used whitespace as the set.

Errors: None.

See also: rtl.sysutils.Trim (1130), RemoveLeadingChars (1150), RemoveTrailingChars (1151), TrimSet (1157), TrimLeftSet (1156), TrimRightSet (1156)

28.3.63 RemoveTrailingChars

Synopsis: Remove any trailing characters in a set from a string

Declaration:
```
procedure RemoveTrailingChars(var S: AnsiString;const CSet: TSysCharset)
```

Visibility: default

Description: `RemoveTrailingChars` removes any trailing characters from `S` that appear in the set `CSet`, i.e. it starts with the last character and works its way to the start of the string. It stops removing characters as soon as a character not in `CSet` is encountered. This is similar in behaviour to TrimRight (1483) which used whitespace as the set.

Errors:

See also: rtl.sysutils.TrimLeft (1130), RemoveLeadingChars (1150), TrimRightSet (1156)

28.3.64 ReverseString

Synopsis: Reverse characters in a string

Declaration:
```
function ReverseString(const AText: String) : String
```

Visibility: default

Description: `ReverseString` returns a string, made up of the characters in string `AText`, in reverse order.

Errors: None.

See also: RandomFrom (1150)

28.3.65 RightBStr

Synopsis: Copy a given number of characters (bytes), counting from the right of a string.

Declaration:
```
function RightBStr(const AText: AnsiString;const AByteCount: Integer)
                         : AnsiString
```

Visibility: default

Description: `RightBStr` returns a string containing the rightmost `AByteCount` bytes from the string `AText`. If `AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned.

Errors: None.

See also: LeftBStr (1146), AnsiRightStr (1137), RightStr (1151), MidBStr (1147)

28.3.66 RightStr

Synopsis: Copy a given number of characters, counting from the right of a string.

Declaration:
```
function RightStr(const AText: AnsiString;const ACount: Integer)
                        : AnsiString
function RightStr(const AText: WideString;const ACount: Integer)
                        : WideString
```

Visibility: default

Description: `RightStr` returns a string containing the rightmost `ACount` characters from the string `AText` . If `ACount` is larger than the length (in characters) of `AText` , only as many characters as available are returned.

Errors: None.

See also: LeftStr (1146), AnsiRightStr (1137), RightBStr (1151), MidStr (1147)

28.3.67 RomanToInt

Synopsis: Convert a string with a Roman number to it's decimal value.

Declaration:
```
function RomanToInt(const S: String) : LongInt
```

Visibility: default

Description: `RomanToInt` returns the decimal equivalent of the Roman numerals in the string `S`. Invalid characters are dropped from `S`. A negative numeral is supported as well.

Errors: None.

See also: IntToRoman (1145), Hex2Dec (1143), Numb2Dec (1148)

28.3.68 RPos

Synopsis: Find last occurrence of substring or character in a string

Declaration:
```
function RPos(c: Char;const S: AnsiString) : Integer;  Overload
function RPos(const Substr: AnsiString;const Source: AnsiString)
                : Integer;  Overload
```

Visibility: default

Description: `RPos` looks in `S` for the character `C` or the string `SubStr`. It starts looking at the end of the string, and searches towards the beginning of the string. If a match is found, it returns the position of the match.

See also: RPosEx (1152)

28.3.69 RPosex

Synopsis: Find last occurrence substring or character in a string, starting at a certain position

Declaration:
```
function RPosEX(C: Char;const S: AnsiString;offs: cardinal) : Integer
                ;  Overload
function RPosex(const Substr: AnsiString;const Source: AnsiString;
                offs: cardinal) : Integer;  Overload
```

Visibility: default

Description: `RPos` looks in `S` for the character `C` or the string `SubStr`. It starts looking at position `Offs`, and searches towards the beginning of the string. If a match is found, it returns the position of the match.

See also: RPos (1152)

28.3.70 SearchBuf

Synopsis: Search a buffer for a certain string.

Declaration:
```
function SearchBuf(Buf: PChar;BufLen: Integer;SelStart: Integer;
                   SelLength: Integer;SearchString: String;
                   Options: TStringSearchOptions) : PChar
function SearchBuf(Buf: PChar;BufLen: Integer;SelStart: Integer;
                   SelLength: Integer;SearchString: String) : PChar
```

Visibility: default

Description: `SearchBuf` searches the buffer `Buf` for the occurrence of `SearchString`. At must `Buflen` characters are searched, and the search is started at `SelStart`+`SelLength`. If a match is found, a pointer to the position of the match is returned. The parameter Options (1131) specifies how the search is conducted. It is a set of the following options:

Table 28.3:

Option	Effect
soDown	Searches forward, starting at the end of the selection. Default is searching up
soMatchCase	Observe case when searching. Default is to ignore case.
soWholeWord	Match only whole words. Default also returns parts of words

The standard constant WordDelimiters (1131) is used to mark the boundaries of words.

The `SelStart` parameter is zero based.

Errors: Buflen must be the real length of the string, no checking on this is performed.

See also: FindPart (1142), ExtractWord (1142), ExtractWordPos (1142), ExtractSubStr (1141), IsWordPresent (1146)

28.3.71 Soundex

Synopsis: Compute the soundex of a string

Declaration:
```
function Soundex(const AText: String;ALength: TSoundexLength) : String
function Soundex(const AText: String) : String
```

Visibility: default

Description: `Soundex` computes a soundex code for `AText`. The resulting code will at most have `ALength` characters. The soundex code is computed according to the US system of soundex computing, which may result in inaccurate results in other languages.

Errors: None.

See also: SoundexCompare (1153), SoundexInt (1154), SoundexProc (1154), SoundexWord (1155), SoundexSimilar (1155)

28.3.72 SoundexCompare

Synopsis: Compare soundex values of 2 strings.

Declaration:
```
function SoundexCompare(const AText: String;const AOther: String;
                        ALength: TSoundexLength) : Integer
function SoundexCompare(const AText: String;const AOther: String)
                        : Integer
```

Visibility: default

Description: `SoundexCompare` computes the soundex codes of `AText` and `AOther` and feeds these to `CompareText`. It will return -1 if the soundex code of `AText` is less than the soundex code of `AOther`, 0 if they are equal, and 1 if the code of `AOther` is larger than the code of `AText`.

Errors: None.

See also: Soundex (1153), SoundexInt (1154), SoundexProc (1154), SoundexWord (1155), SoundexSimilar (1155)

28.3.73 SoundexInt

Synopsis: Soundex value as an integer.

Declaration:
```
function SoundexInt(const AText: String;ALength: TSoundexIntLength)
                    : Integer
function SoundexInt(const AText: String) : Integer
```

Visibility: default

Description: `SoundexInt` computes the Soundex (1153) code (with length `ALength`, default 4) of `AText`, and converts the code to an integer value.

Errors: None.

See also: Soundex (1153), SoundexCompare (1153), SoundexProc (1154), SoundexWord (1155), SoundexSimilar (1155)

28.3.74 SoundexProc

Synopsis: Default AnsiResemblesText implementation.

Declaration:
```
function SoundexProc(const AText: String;const AOther: String) : Boolean
```

Visibility: default

Description: `SoundexProc` is the standard implementation for the AnsiResemblesText (1136) procedure: By default, `AnsiResemblesProc` is set to this function. It compares the soundex codes of `AOther` and `AText` and returns `True` if they are equal, or `False` if they are not.

Errors: None.

See also: Soundex (1153), SoundexCompare (1153), SoundexInt (1154), SoundexWord (1155), SoundexSimilar (1155)

28.3.75 SoundexSimilar

Synopsis: Check whether 2 strings have equal soundex values

Declaration:
```
function SoundexSimilar(const AText: String;const AOther: String;
                        ALength: TSoundexLength) : Boolean
function SoundexSimilar(const AText: String;const AOther: String)
                        : Boolean
```

Visibility: default

Description: `SoundexSimilar` returns `True` if the soundex codes (with length `ALength`) of `AText` and `AOther` are equal, and `False` if they are not.

Errors: None.

See also: Soundex (1153), SoundexCompare (1153), SoundexInt (1154), SoundexProc (1154), SoundexWord (1155), Soundex (1153)

28.3.76 SoundexWord

Synopsis: Calculate a word-sized soundex value

Declaration:
```
function SoundexWord(const AText: String) : Word
```

Visibility: default

Description: `SoundexInt` computes the Soundex (1153) code (with length 4) of `AText` , and converts the code to a word-sized value.

Errors: None.

See also: Soundex (1153), SoundexCompare (1153), SoundexInt (1154), SoundexProc (1154), SoundexSimilar (1155)

28.3.77 StringsReplace

Synopsis: Replace occurrences of a set of strings to another set of strings

Declaration:
```
function StringsReplace(const S: String;OldPattern: Array of String;
                        NewPattern: Array of String;Flags: TReplaceFlags)
                        : String
```

Visibility: default

Description: `StringsReplace` scans `S` for the occurrence of one of the strings in `OldPattern` and replaces it with the corresponding string in `NewPattern`. It takes into account `Flags`, which has the same meaning as in StringReplace (1461).

Corresponding strings are matched by location: the N-th string in `OldPattern` is replaced by the N-th string in `NewPattern`. Note that this means that the number of strings in both arrays must be the same.

Errors: If the number of strings in both arrays is different, then an exception is raised.

See also: #rtl.sysutils.StringReplace (1461), #rtl.sysutils.TReplaceFlags (1366)

28.3.78 StuffString

Synopsis: Replace part of a string with another string.

Declaration:
```
function StuffString(const AText: String;AStart: Cardinal;
                     ALength: Cardinal;const ASubText: String) : String
```

Visibility: default

Description: `StuffString` returns a copy of `AText` with the segment starting at `AStart` with length `ALength`, replaced with the string `ASubText`. Basically it deletes the segment of `Atext` and inserts the new text in it's place.

Errors: No checking on the validity of the AStart and ALength parameters is done. Providing invalid values may result in access violation errors.

See also: FindPart (1142), DelChars (1140), DelSpace (1140), ExtractSubStr (1141), DupeString (1141)

28.3.79 Tab2Space

Synopsis: Convert tab characters to a number of spaces

Declaration:
```
function Tab2Space(const S: String;Numb: Byte) : String
```

Visibility: default

Description: `Tab2Space` returns a copy of `S` with all tab characters (ASCII character 9) converted to `Numb` spaces.

Errors: None.

See also: StuffString (1156), FindPart (1142), ExtractWord (1142), DelChars (1140), DelSpace (1140), DelSpace1 (1140), DupeString (1141)

28.3.80 TrimLeftSet

Synopsis: Remove any leading characters in a set from a string and returns the result

Declaration:
```
function TrimLeftSet(const S: String;const CSet: TSysCharSet) : String
```

Visibility: default

Description: `TrimLeftSet` performs the same action as RemoveLeadingChars (1150), but returns the resulting string.

Errors: None.

See also: rtl.sysutils.TrimLeft (1130), RemoveLeadingChars (1150), RemoveTrailingChars (1151), RemovePadChars (1150), TrimSet (1157), TrimRightSet (1156)

28.3.81 TrimRightSet

Synopsis: Remove any trailing characters in a set from a string and returns the result

Declaration:
```
function TrimRightSet(const S: String;const CSet: TSysCharSet) : String
```

Visibility: default

Description: `TrimLeftSet` performs the same action as RemoveTrailingChars (1151), but returns the resulting string.

Errors: None.

See also: rtl.sysutils.TrimRight (1130), RemoveLeadingChars (1150), RemoveTrailingChars (1151), RemovePadChars (1150), TrimSet (1157), TrimLeftSet (1156)

28.3.82 TrimSet

Synopsis: Remove any leading or trailing characters in a set from a string and returns the result

Declaration:
```
function TrimSet(const S: String;const CSet: TSysCharSet) : String
```

Visibility: default

Description: `TrimSet` performs the same action as RemovePadChars (1150), but returns the resulting string.

Errors: None.

See also: rtl.sysutils.Trim (1130), RemoveLeadingChars (1150), RemoveTrailingChars (1151), RemovePadChars (1150), TrimRightSet (1156), TrimLeftSet (1156)

28.3.83 WordCount

Synopsis: Count the number of words in a string.

Declaration:
```
function WordCount(const S: String;const WordDelims: TSysCharSet)
                         : Integer
```

Visibility: default

Description: `WordCount` returns the number of words in the string `S`. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`.

The pre-defined StdWordDelims (1131) constant can be used for the `WordDelims` argument.

Errors: None.

See also: WordPosition (1157), StdWordDelims (1131), ExtractWord (1142), ExtractWordPos (1142)

28.3.84 WordPosition

Synopsis: Search position of Nth word in a string.

Declaration:
```
function WordPosition(const N: Integer;const S: String;
                            const WordDelims: TSysCharSet) : Integer
```

Visibility: default

Description: `WordPosition` returns the position (in characters) of the `N`-th word in the string `S`. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`. If `N` is out of range, zero is returned.

The pre-defined StdWordDelims (1131) constant can be used for the `WordDelims` argument.

Errors: None

See also: WordCount (1157), StdWordDelims (1131), ExtractWord (1142), ExtractWordPos (1142)

28.3.85 XorDecode

Synopsis: Decode a string encoded with XorEncode (1158)

Declaration:
```
function XorDecode(const Key: String;const Source: String) : String
```

Visibility: default

Description: `XorDecode` decodes `Source` and returns the original string that was encrypted using XorEncode (1158) with key `Key`. If a different key is used than the key used to encode the string, the result will be unreadable.

Errors: If the string `Source` is not a valid XorEncode result (e.g. contains non-numerical characters), then a `EConversionError` exception will be raised.

See also: XorEncode (1158), XorString (1158)

28.3.86 XorEncode

Synopsis: Encode a string by XOR-ing its characters using characters of a given key, representing the result as hex values.

Declaration:
```
function XorEncode(const Key: String;const Source: String) : String
```

Visibility: default

Description: `XorEncode` encodes the string `Source` by XOR-ing each character in `Source` with the corresponding character in `Key` (repeating `Key` as often as necessary) and representing the resulting ASCII code as a hexadecimal number (of length 2). The result is therefore twice as long as the original string, and every 2 bytes represent an ASCII code.

Feeding the resulting string with the same key `Key` to the XorDecode (1158) function will result in the original `Source` string.

This function can be used e.g. to trivially encode a password in a configuration file.

Errors: None.

See also: XorDecode (1158), XorString (1158), Hex2Dec (1143)

28.3.87 XorString

Synopsis: Encode a string by XOR-ing its characters using characters of a given key.

Declaration:
```
function XorString(const Key: ShortString;const Src: ShortString)
                         : ShortString
```

Visibility: default

Description: `XorString` encodes the string `Src` by XOR-ing each character in `Source` with the corresponding character in `Key`, repeating `Key` as often as necessary. The resulting string may contain unreadable characters and may even contain null characters. For this reason it may be a better idea to use the XorEncode (1158) function instead, which will representing each resulting ASCII code as a hexadecimal number (of length 2).

Feeding the result again to `XorString` with the same `Key`, will result in the original string `Src`.

Errors: None.

See also: XorEncode (1158), XorDecode (1158)

Chapter 29

Reference for unit 'System'

29.1 Miscellaneous functions

Functions that do not belong in one of the other categories.

Table 29.1:

Name	Description
Assert (1209)	Conditionally abort program with error
Break (1215)	Abort current loop
Continue (1223)	Next cycle in current loop
Exclude (1233)	Exclude an element from a set
Exit (1234)	Exit current function or procedure
Include (1251)	Include an element into a set
LongJmp (1263)	Jump to execution point
Ord (1292)	Return ordinal value of enumerated type
Pred (1296)	Return previous value of ordinal type
SetJmp (1310)	Mark execution point for jump
SizeOf (1315)	Return size of variable or type
Succ (1321)	Return next value of ordinal type

29.2 Operating System functions

Functions that are connected to the operating system.

29.3 String handling

All things connected to string handling.

29.4 Mathematical routines

Functions connected to calculating and coverting numbers.

Table 29.2:

Name	Description
Chdir (1216)	Change working directory
Getdir (1243)	Return current working directory
Halt (1247)	Halt program execution
Paramcount (1293)	Number of parameters with which program was called
Paramstr (1294)	Retrieve parameters with which program was called
Mkdir (1264)	Make a directory
Rmdir (1304)	Remove a directory
Runerror (1307)	Abort program execution with error condition

Table 29.3:

Name	Description
BinStr (1213)	Construct binary representation of integer
Chr (1216)	Convert ASCII code to character
Concat (1222)	Concatenate two strings
Copy (1224)	Copy part of a string
Delete (1227)	Delete part of a string
HexStr (1247)	Construct hexadecimal representation of integer
Insert (1255)	Insert one string in another
Length (1260)	Return length of string
Lowercase (1264)	Convert string to all-lowercase
OctStr (1267)	Construct octal representation of integer
Pos (1295)	Calculate position of one string in another
SetLength (1311)	Set length of a string
SetString (1312)	Set contents and length of a string
Str (1318)	Convert number to string representation
StringOfChar (1319)	Create string consisting of a number of characters
Upcase (1329)	Convert string to all-uppercase
Val (1331)	Convert string to number

29.5 Memory management functions

Functions concerning memory issues.

29.6 File handling functions

Functions concerning input and output from and to file.

29.7 Overview

The system unit contains the standard supported functions of Free Pascal. It is the same for all platforms. Basically it is the same as the system unit provided with Borland or Turbo Pascal.

Functions are listed in alphabetical order. Arguments of functions or procedures that are optional are put between square brackets.

Table 29.4:

Name	Description
Abs (1205)	Calculate absolute value
Arctan (1208)	Calculate inverse tangent
Cos (1224)	Calculate cosine of angle
Dec (1225)	Decrease value of variable
Exp (1235)	Exponentiate
Frac (1241)	Return fractional part of floating point value
Hi (1248)	Return high byte/word of value
Inc (1250)	Increase value of variable
Int (1256)	Calculate integer part of floating point value
Ln (1261)	Calculate logarithm
Lo (1262)	Return low byte/word of value
Odd (1268)	Is a value odd or even ?
Pi (1294)	Return the value of pi
Power (1159)	Raise float to integer power
Random (1297)	Generate random number
Randomize (1298)	Initialize random number generator
Round (1305)	Round floating point value to nearest integer number
Sin (1315)	Calculate sine of angle
Sqr (1317)	Calculate the square of a value
Sqrt (1318)	Calculate the square root of a value
Swap (1322)	Swap high and low bytes/words of a variable
Trunc (1327)	Truncate a floating point value

The pre-defined constants and variables are listed in the first section. The second section contains an overview of all functions, grouped by functionality, and the last section contains the supported functions and procedures.

29.8 Constants, types and variables

29.8.1 Constants

```
AbstractErrorProc : TAbstractErrorProc = nil
```

If set, the `AbstractErrorProc` constant is used when an abstract error occurs. If it is not set, then the standard error handling is done: A stack dump is performed, and the program exits with error code 211.

The SysUtils unit sets this procedure and raises an exception in its handler.

```
AllFilesMask = '*'
```

`AllFilesMask` is the wildcard that can be used to return all files in a directory. On windows and dos based systems, this will be '*.*', while for unix systems, this will be '*'.

```
AllowDirectorySeparators : Set of Char = ['\','/']
```

`AllowDirectorySeparators` is the set of characters which are considered directory separators by the RTL units. By default, this is set to the most common directory separators: forward slash and backslash, so routines will work in a cross-platform manner, no matter which character was used:

Table 29.5:

Name	Description
Addr (1206)	Return address of variable
Assigned (1210)	Check if a pointer is valid
CompareByte (1217)	Compare 2 memory buffers byte per byte
CompareChar (1218)	Compare 2 memory buffers byte per byte
CompareDWord (1220)	Compare 2 memory buffers byte per byte
CompareWord (1221)	Compare 2 memory buffers byte per byte
CSeg (1225)	Return code segment
Dispose (1227)	Free dynamically allocated memory
DSeg (1228)	Return data segment
FillByte (1237)	Fill memory region with 8-bit pattern
Fillchar (1238)	Fill memory region with certain character
FillDWord (1239)	Fill memory region with 32-bit pattern
Fillword (1239)	Fill memory region with 16-bit pattern
Freemem (1242)	Release allocated memory
Getmem (1244)	Allocate new memory
GetMemoryManager (1245)	Return current memory manager
High (1249)	Return highest index of open array or enumerated
IsMemoryManagerSet (1259)	Is the memory manager set
Low (1263)	Return lowest index of open array or enumerated
Move (1265)	Move data from one location in memory to another
MoveChar0 (1265)	Move data till first zero character
New (1266)	Dynamically allocate memory for variable
Ofs (1268)	Return offset of variable
Ptr (1296)	Combine segment and offset to pointer
ReAllocMem (1301)	Resize a memory block on the heap
Seg (1309)	Return segment
SetMemoryManager (1311)	Set a memory manager
Sptr (1317)	Return current stack pointer
SSeg (1318)	Return stack segment register value

```
AllowDirectorySeparators : set of char = ['\','/'];
```

If a more strict behaviour is desired, then `AllowDirectorySeparators` can be set to the only character allowed on the current operating system, and all RTL routines that handle filenames (splitting filenames, extracting parts of the filename and so on) will use that character only.

```
AllowDriveSeparators : Set of Char = []
```

`AllowDriveSeparators` are the characters which are considered to separate the drive part from the directory part in a filename. This will be an empty set on systems that do not support drive letters. Other systems (dos, windows and OS/2) will have the colon (:) character as the only member of this set.

```
AssertErrorProc : TAssertErrorProc = @SysAssert
```

If set, the `AbstractErrorProc` constant is used when an assert error occurs. If it is not set, then the standard error handling is done: The assertion error message is printed, together with the location of the assertion, and A stack dump is performed, and the program exits with error code 227.

The SysUtils unit sets this procedure and raises an exception in its handler.

Table 29.6:

Name	Description
Append (1208)	Open a file in append mode
Assign (1210)	Assign a name to a file
Blockread (1213)	Read data from a file into memory
Blockwrite (1214)	Write data from memory to a file
Close (1217)	Close a file
Eof (1231)	Check for end of file
Eoln (1232)	Check for end of line
Erase (1232)	Delete file from disk
Filepos (1236)	Position in file
Filesize (1236)	Size of file
Flush (1240)	Write file buffers to disk
IOresult (1258)	Return result of last file IO operation
Read (1298)	Read from file into variable
Readln (1299)	Read from file into variable and goto next line
Rename (1302)	Rename file on disk
Reset (1302)	Open file for reading
Rewrite (1303)	Open file for writing
Seek (1307)	Set file position
SeekEof (1308)	Set file position to end of file
SeekEoln (1309)	Set file position to end of line
SetTextBuf (1312)	Set size of file buffer
Truncate (1327)	Truncate the file at position
Write (1334)	Write variable to file
WriteLn (1335)	Write variable to file and append newline

```
BackTraceStrFunc : TBackTraceStrFunc = @SysBackTraceStr
```

This handler is called to get a standard format for the backtrace routine.

```
CtrlZMarksEOF : Boolean = false
```

`CtrlZMarksEOF` indicates whether on this system, an `CTRL-Z` character (ordinal 26) in a file marks the end of the file. This is `False` on most systems except on DOS.

To get DOS-compatible behaviour, this constant can be set to `True`

```
DefaultStackSize = 32768
```

Default size for a new thread's stack (32k by default).

```
DefaultTextLineBreakStyle : TTextLineBreakStyle = tlbsLF
```

`DefaultTextLineBreakStyle` contains the default OS setting for the TTextLineBreakStyle (1197) type. It is initizliaed by the system unit, and is used to determine the default line ending when writing to text files.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DirectorySeparator = '/'
```

DirectorySeparator is the character used by the current operating system to separate directory parts in a pathname. This constant is system dependent, and should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DriveSeparator = '/'
```

On systems that support driveletters, the DriveSeparator constant denotes the character that separates the drive indicator from the directory part in a filename path.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
Erroraddr : pointer = nil
```

Address where the last error occurred.

```
Errorcode : Word = 0
```

Last error code.

```
ErrorProc : TErrorProc = nil
```

If set, the ErrorProc constant is used when a run-time error occurs. If it is not set, then the standard error handling is done: a stack dump is performed, and the program exits with the indicated error code.

The SysUtils unit sets this procedure and raises an exception in its handler.

```
ExceptProc : TExceptProc = nil
```

This constant points to the current exception handling procedure. This routine is called when an unhandled exception occurs, i.e. an exception that is not stopped by a except block.

If the handler is not set, the RTL will emit a run-time error 217 when an unhandler exception occurs.

It is set by the sysutils (1350) unit.

```
ExitProc : pointer = nil
```

Exit procedure pointer.

```
ExtensionSeparator = '.'
```

ExtensionSeparator is the character which separates the filename from the file extension. On all current platforms, this is the . (dot) character. All RTL filename handling routines use this constant.

```
E_NOINTERFACE = hresult ( $80004002 )
```

Interface call result: Error: not an interface

```
E_NOTIMPL = hresult ( $80004001 )
```

Interface call result: Interface not implemented

```
E_UNEXPECTED = hresult ( $8000FFFF )
```

Interface call result: Unexpected error

```
Filemode : Byte = 2
```

Default file mode for untyped files.

```
FileNameCaseSensitive : Boolean = true
```

`FileNameCaseSensitive` is `True` if case is important when using filenames on the current OS. In this case, the OS will treat files with different cased names as different files. Note that this may depend on the filesystem: Unix operating systems that access a DOS or Windows partition will have this constant set to true, but when writing to the DOS partition, the casing is ignored.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
float_flag_denormal = 2
```

IEC/IEEE floating-point exception flag: ?

```
float_flag_divbyzero = 4
```

IEC/IEEE floating-point exception flag: Division by zero error

```
float_flag_inexact = 32
```

IEC/IEEE floating-point exception flag: ?

```
float_flag_invalid = 1
```

IEC/IEEE floating-point exception flag: Invalid operation error

```
float_flag_overflow = 8
```

IEC/IEEE floating-point exception flag: Overflow error

```
float_flag_underflow = 16
```

IEC/IEEE floating-point exception flag: Underflow error

```
fmAppend = $D7B4
```

File mode: File is open for writing, appending to the end.

```
fmClosed = $D7B0
```

File mode: File is closed.

```
fmInOut = $D7B3
```

File mode: File is open for reading and writing.

```
fmInput = $D7B1
```

File mode: File is open for reading.

```
fmOutput = $D7B2
```

File mode: File is open for writing.

```
fpc_in_abs_real = 127
```

FPC compiler internal procedure index: abs (real)

```
fpc_in_addr_x = 42
```

FPC compiler internal procedure index: addr

```
fpc_in_arctan_real = 130
```

FPC compiler internal procedure index: arctan (real)

```
fpc_in_assert_x_y = 41
```

FPC compiler internal procedure index: assert

```
fpc_in_assigned_x = 19
```

FPC compiler internal procedure index: assigned

```
fpc_in_bitsizeof_x = 61
```

FPC compiler internal procedure index: bitsizeof

```
fpc_in_break = 39
```

FPC compiler internal procedure index: break

```
fpc_in_chr_byte = 7
```

FPC compiler internal procedure index: chr

```
fpc_in_concat_x = 18
```

FPC compiler internal procedure index: concat

```
fpc_in_const_abs = 101
```

FPC compiler internal procedure index: abs

```
fpc_in_const_odd = 102
```

FPC compiler internal procedure index: sqr

```
fpc_in_const_ptr = 103
```

FPC compiler internal procedure index: sqr

```
fpc_in_const_sqr = 100
```

FPC compiler internal procedure index: sqr

```
fpc_in_const_swap_long = 105
```

FPC compiler internal procedure index: swap (long)

```
fpc_in_const_swap_qword = 108
```

FPC compiler internal procedure index: swap (qword)

```
fpc_in_const_swap_word = 104
```

FPC compiler internal procedure index: swap (word)

```
fpc_in_continue = 40
```

FPC compiler internal procedure index: continue

```
fpc_in_copy_x = 49
```

FPC compiler internal procedure index: copy

```
fpc_in_cos_real = 125
```

FPC compiler internal procedure index: cos (real)

```
fpc_in_cycle = 52
```

FPC compiler internal procedure index: cycle

```
fpc_in_dec_x = 36
```

FPC compiler internal procedure index: dec

```
fpc_in_dispose_x = 47
```

FPC compiler internal procedure index: dispose

```
fpc_in_exclude_x_y = 38
```

FPC compiler internal procedure index: exclude

```
fpc_in_exit = 48
```

FPC compiler internal procedure index: exit

```
fpc_in_exp_real = 124
```

FPC internal compiler routine: in_exp_real

```
fpc_in_fillchar_x = 55
```

FPC internal compiler routine: in_fillchar_x

```
fpc_in_finalize_x = 45
```

FPC compiler internal procedure index: finalize

```
fpc_in_frac_real = 122
```

FPC internal compiler routine: in_frac_real

```
fpc_in_get_caller_addr = 57
```

FPC internal compiler routine: in_get_caller_addr

```
fpc_in_get_caller_frame = 58
```

FPC internal compiler routine: in_get_caller_frame

```
fpc_in_get_frame = 56
```

FPC internal compiler routine: in_get_frame

```
fpc_in_high_x = 28
```

FPC compiler internal procedure index: high

```
fpc_in_hi_long = 4
```

FPC compiler internal procedure index: hi (long)

```
fpc_in_hi_qword = 107
```

FPC compiler internal procedure index: hi (qword)

```
fpc_in_hi_word = 2
```

FPC compiler internal procedure index: hi (word)

```
fpc_in_include_x_y = 37
```

FPC compiler internal procedure index: include

```
fpc_in_inc_x = 35
```

FPC compiler internal procedure index: inc

```
fpc_in_initialize_x = 50
```

FPC compiler internal procedure index: initialize

```
fpc_in_int_real = 123
```

FPC internal compiler routine: in_int_real

```
fpc_in_leave = 51
```

FPC compiler internal procedure index: leave

```
fpc_in_length_string = 6
```

FPC compiler internal procedure index: length

```
fpc_in_ln_real = 131
```

FPC compiler internal procedure index: ln (real)

```
fpc_in_low_x = 27
```

FPC compiler internal procedure index: low

```
fpc_in_lo_long = 3
```

FPC compiler internal procedure index: lo (long)

```
fpc_in_lo_qword = 106
```

FPC compiler internal procedure index: lo (qword)

```
fpc_in_lo_word = 1
```

FPC compiler internal procedure index: lo (word)

```
fpc_in_mmx_pcmpeqb = 200
```

FPC compiler internal procedure index: MMX

```
fpc_in_mmx_pcmpeqd = 202
```

FPC compiler internal procedure index: MMX

```
fpc_in_mmx_pcmpeqw = 201
```

FPC compiler internal procedure index: MMX

```
fpc_in_mmx_pcmpgtb = 203
```

FPC compiler internal procedure index: MMX

```
fpc_in_mmx_pcmpgtd = 205
```

FPC compiler internal procedure index: MMX

```
fpc_in_mmx_pcmpgtw = 204
```

FPC compiler internal procedure index: MMX

```
fpc_in_move_x = 54
```

FPC internal compiler routine: in_move_x

```
fpc_in_new_x = 46
```

FPC compiler internal procedure index: new

```
fpc_in_ofs_x = 21
```

FPC compiler internal procedure index: ofs

```
fpc_in_ord_x = 5
```

FPC compiler internal procedure index: ord

```
fpc_in_pack_x_y_z = 59
```

FPC compiler internal procedure index: pack

```
fpc_in_pi_real = 126
```

FPC internal compiler routine: in_pi_real

```
fpc_in_pred_x = 30
```

FPC compiler internal procedure index: pred

```
fpc_in_prefetch_var = 109
```

FPC compiler internal procedure index: prefetch

```
fpc_in_readln_x = 17
```

FPC compiler internal procedure index: readln

```
fpc_in_read_x = 16
```

FPC compiler internal procedure index: read

```
fpc_in_reset_typedfile = 32
```

FPC compiler internal procedure index: reset

```
fpc_in_reset_x = 25
```

FPC compiler internal procedure index: reset

```
fpc_in_rewrite_typedfile = 33
```

FPC compiler internal procedure index: rewrite

```
fpc_in_rewrite_x = 26
```

FPC compiler internal procedure index: rewrite

```
fpc_in_round_real = 121
```

FPC internal compiler routine: in_round_real

```
fpc_in_seg_x = 29
```

FPC compiler internal procedure index: seg

```
fpc_in_setlength_x = 44
```

FPC compiler internal procedure index: setlength

```
fpc_in_settextbuf_file_x = 34
```

FPC compiler internal procedure index: settextbuf

```
fpc_in_sin_real = 132
```

FPC compiler internal procedure index: sin (real)

```
fpc_in_sizeof_x = 22
```

FPC compiler internal procedure index: sizeof

```
fpc_in_slice = 53
```

FPC internal compiler routine: in_slice

```
fpc_in_sqrt_real = 129
```

FPC compiler internal procedure index: sqrt (real)

```
fpc_in_sqr_real = 128
```

FPC compiler internal procedure index: sqr (real)

```
fpc_in_str_x_string = 20
```

FPC compiler internal procedure index: str

```
fpc_in_succ_x = 31
```

FPC compiler internal procedure index: succ

```
fpc_in_trunc_real = 120
```

FPC internal compiler routine: in_trunc_real

```
fpc_in_typeinfo_x = 43
```

FPC compiler internal procedure index: typeinfo

```
fpc_in_typeof_x = 23
```

FPC compiler internal procedure index: typeof

```
fpc_in_unpack_x_y_z = 60
```

FPC compiler internal procedure index: unpack

```
fpc_in_val_x = 24
```

FPC compiler internal procedure index: val

```
fpc_in_writeln_x = 15
```

FPC compiler internal procedure index: writeln

```
fpc_in_write_x = 14
```

FPC compiler internal procedure index: write

```
growheapsize1 : PtrInt = 256 * 1024
```

Grow rate for block less than 256 Kb.

```
growheapsize2 : PtrInt = 1024 * 1024
```

Grow rate for block larger than 256 Kb.

```
growheapsizesmall : PtrInt = 32 * 1024
```

Fixed size small blocks grow rate

```
InitProc : Pointer = nil
```

InitProc is a routine that can be called after all units were initialized. It can be set by units to execute code that can be initialized after all units were initialized.

Remark: When setting the value of InitProc, the previous value should always be saved, and called when the installed initialization routine has finished executing.

```
IsMultiThread : longbool = false
```

Indicates whether more than one thread is running in the application.

```
LFNSupport = true
```

LFNSupport determines whether the current OS supports long file names, i.e. filenames that are not of the form 8.3 as on ancient DOS systems. If the value of this constant is True then long filenames are supported. If it is false, then not.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
LineEnding = #10
```

LineEnding is a constant which contains the current line-ending character. This character is system dependent, and is initialized by the system. It should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
maxExitCode = 255
```

maxExitCode is the maximum value for the Halt (1247) call.

```
maxint = maxsmallint
```

Maximum integer value.

```
MaxKeptOSChunks : DWord = 4
```

MaxKeptOSChunks tells the heap manager how many free chunks of OS-allocated memory it should keep in memory. When freeing memory, it can happen that a memory block obtained from the OS is completely free. If more than MaxKeptOSChunks such blocks are free, then the heap manager will return them to the OS, to reduce memory requirements.

```
maxLongint = $7fffffff
```

Maximum longint value.

```
MaxPathLen = 4096
```

This constant is system dependent.

```
MaxSIntValue = High ( ValSInt )
```

Maximum String-size value.

```
maxSmallint = 32767
```

Maximum smallint value.

```
MaxUIntValue = High ( ValUInt )
```

Maximum unsigned integer value.

```
Max_Frame_Dump : Word = 8
```

Maximum number of frames to show in error frame dump.

```
ModuleIsCpp : Boolean = false
```

`ModuleIsCpp` is always false for FPC programs, it is provided for Delphi compatibility only.

```
ModuleIsLib : Boolean = false
```

`ModuleIsLib` is set by the compiler when linking a library, program or package, and determines whether the current module is a library (or package) (`True`) or program (`False`).

```
ModuleIsPackage : Boolean = false
```

`ModuleIsLib` is set by the compiler when linking a library, program or package, and determines whether the current module is a package (`True`) or a library or program (`False`).

```
PathSeparator = ':'
```

`PathSeparator` is the character used commonly on the current operating system to separate paths in a list of paths, such as the `PATH` environment variable.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
RaiseMaxFrameCount : LongInt = 16
```

Maximum number of frames to include in TExceptObject (1192)

```
RaiseProc : TExceptProc = nil
```

Procedure to raise an exception.

```
RT_ACCELERATOR = MAKEINTRESOURCE ( 9 )
```

Constant identifying an accelerator resouce

```
RT_BITMAP = MAKEINTRESOURCE ( 2 )
```

Constant identifying a bitmap resource

```
RT_CURSOR = MAKEINTRESOURCE ( 1 )
```

Constant identifying a cursor resource

```
RT_DIALOG = MAKEINTRESOURCE ( 5 )
```

Constant identifying a dialog resource

```
RT_FONT = MAKEINTRESOURCE ( 8 )
```

Constant identifying a font resource

```
RT_FONTDIR = MAKEINTRESOURCE ( 7 )
```

Constant identifying a font directory resource

```
RT_GROUP_CURSOR = MAKEINTRESOURCE ( 12 )
```

Constant identifying a group cursor resource

```
RT_GROUP_ICON = MAKEINTRESOURCE ( 13 )
```

Constant identifying a group icon resource

```
RT_ICON = MAKEINTRESOURCE ( 3 )
```

Constant identifying an icon resource

```
RT_MENU = MAKEINTRESOURCE ( 4 )
```

Constant identifying a menu resource

```
RT_MESSAGETABLE = MAKEINTRESOURCE ( 11 )
```

Constant identifying a message data resource

```
RT_RCDATA = MAKEINTRESOURCE ( 10 )
```

Constant identifying a binary data resource

```
RT_STRING = MAKEINTRESOURCE ( 6 )
```

Constant identifying a string table resource

```
RT_VERSION = MAKEINTRESOURCE ( 16 )
```

Constant identifying a version info resource

```
RuntimeErrorExitCodes : Array[TRuntimeError] of Byte = (0,203,204,200,201,215,207,20
```

This array is used by the Error (1233) routine to to convert a TRuntimeError (1196) enumeration type to a process exit code.

```
SafeCallErrorProc : TSafeCallErrorProc = nil
```

SafeCallErrorProc is a Handler called in case of a safecall calling convention error. Error is the error number (passed by the Windows operating system) and Addr is the address where the error occurred.

```
SIGSTKSZ = 40960
```

```
sLineBreak = LineEnding
```

sLineBreak is an alias for LineEnding (1173) and is supplied for Delphi compatibility.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
StackError : Boolean = false
```

Indicate whether there was a stack error.

```
StdErrorHandle = 2
```

Value of the OS handle for the standard error-output file.

```
StdInputHandle = 0
```

Value of the OS handle for the standard input file.

```
StdOutputHandle = 1
```

Value of the OS handle for the standard output file.

```
S_FALSE = 1
```

Interface call result: Not OK

```
S_OK = 0
```

Interface call result: OK

```
ThreadingAlreadyUsed : Boolean = false
```

Internal constant for the threading system. Don't use.

```
UnusedHandle = -1
```

Value indicating an unused file handle (as reported by the OS).

```
VarAddRefProc : procedure(var v: tvardata) = nil
```

Callback to increase reference count of a variant.

```
varany = $101
```

Variant type: Any

```
vararray = $2000
```

Variant type: variant Array

```
varboolean = 11
```

Variant type: Boolean type

```
varbyref = $4000
```

Variant type: By reference

```
varbyte = 17
```

Variant type: Byte (8 bit)

```
VarClearProc : procedure(var v: tvardata) = nil
```

Callback to clear a variant.

```
VarCopyProc : procedure(var d: tvardata;const s: tvardata) = nil
```

Callback to copy a variant

```
varcurrency = 6
```

Variant type: Currency

```
vardate = 7
```

Variant type: Date

```
vardecimal = 14
```

Variant type: Decimal (BCD)

```
vardispatch = 9
```

Variant type: dispatch interface

```
vardouble = 5
```

Variant type: Double float

```
varempty = 0
```

Variant type: `Empty` variant

```
varerror = 10
```

Variant type: Error type

```
varint64 = 20
```

Variant type: Integer (64-Bit)

```
varinteger = 3
```

Variant type: Integer (32-bit)

```
varlongword = 19
```

Variant type: Word (32 bit)

```
varnull = 1
```

Variant type: Null (1267) variant

```
varolestr = 8
```

Variant type: OLE string (widestring)

```
varqword = 21
```

Variant type: Word (64-bit)

```
varrecord = 36
```

```
varshortint = 16
```

Variant type: Shortint (16 bit)

```
varsingle = 4
```

Variant type: Single float

```
varsmallint = 2
```

Variant type: smallint (8 bit)

```
varstrarg = $48
```

Variant type: String

```
varstring = $100
```

Variant type: String

```
VarToLStrProc : procedure(var d: AnsiString;const s: tvardata) = nil
```

Callback to convert a variant to a ansistring.

```
VarToWStrProc : procedure(var d: WideString;const s: tvardata) = nil
```

Callback to convert a variant to a widestring.

```
vartypemask = $fff
```

Variant type: Mask to extract type

```
varunknown = 13
```

Variant type: Unknown

```
varvariant = 12
```

Variant type: Variant (arrays only)

```
varword = 18
```

Variant type: Word (16 bit)

```
varword64 = varqword
```

Variant type: Word (64-bit)

```
vmtAfterConstruction = vmtMethodStart + sizeof ( pointer ) * 5
```

VMT Layout: ?

```
vmtAutoTable = vmtParent + sizeof ( pointer ) * 7
```

VMT layout: ?

```
vmtBeforeDestruction = vmtMethodStart + sizeof ( pointer ) * 6
```

VMT Layout: ?

```
vmtClassName = vmtParent + sizeof ( pointer )
```

VMT Layout: location of class name.

```
vmtDefaultHandler = vmtMethodStart + sizeof ( pointer ) * 4
```

VMT Layout: ?

```
vmtDefaultHandlerStr = vmtMethodStart + sizeof ( pointer ) * 7
```

VMT Layout: ?

```
vmtDestroy = vmtMethodStart
```

VMT Layout: Location of destructor pointer.

```
vmtDynamicTable = vmtParent + sizeof ( pointer ) * 2
```

VMT Layout: location of dynamic methods table.

```
vmtFieldTable = vmtParent + sizeof ( pointer ) * 4
```

VMT Layout: Location of fields table.

```
vmtFreeInstance = vmtMethodStart + sizeof ( pointer ) * 2
```

VMT Layout: location of FreeInstance method.

```
vmtInitTable = vmtParent + sizeof ( pointer ) * 6
```

VMT Layout: ?

```
vmtInstanceSize = 0
```

VMT Layout: Location of class instance size in VMT

```
vmtIntfTable = vmtParent + sizeof ( pointer ) * 8
```

VMT layout: Interface table

```
vmtMethodStart = vmtParent + sizeof ( pointer ) * 10
```

VMT layout: start of method table.

```
vmtMethodTable = vmtParent + sizeof ( pointer ) * 3
```

VMT Layout: Method table start.

```
vmtMsgStrPtr = vmtParent + sizeof ( pointer ) * 9
```

VMT layout: message strings table.

```
vmtNewInstance = vmtMethodStart + sizeof ( pointer )
```

VMT Layout: location of NewInstance method.

```
vmtParent = sizeof ( ptrint ) * 2
```

VMT Layout: location of pointer to parent VMT.

```
vmtSafeCallException = vmtMethodStart + sizeof ( pointer ) * 3
```

VMT Layout: ?

```
vmtTypeInfo = vmtParent + sizeof ( pointer ) * 5
```

VMT Layout: Location of class type information.

```
vtAnsiString = 11
```

TVarRec type: Ansistring

```
vtBoolean = 1
```

TVarRec type: Boolean

```
vtChar = 2
```

TVarRec type: Char

```
vtClass = 8
```

TVarRec type: Class type

```
vtCurrency = 12
```

TVarRec type: Currency

```
vtExtended = 3
```

TVarRec type: Extended

```
vtInt64 = 16
```

TVarRec type: Int64 (signed 64-bit integer)

```
vtInteger = 0
```

TVarRec type: Integer

```
vtInterface = 14
```

TVarRec type: Interface

```
vtObject = 7
```

TVarRec type: Object instance

```
vtPChar = 6
```

TVarRec type: PChar

```
vtPointer = 5
```

TVarRec type: pointer

```
vtPWideChar = 10
```

TVarRec type: PWideChar

```
vtQWord = 17
```

TVarRec type: QWord (unsigned 64-bit integer)

```
vtString = 4
```

TVarRec type: String

```
vtVariant = 13
```

TVarRec type: Variant

```
vtWideChar = 9
```

TVarRec type: Widechar

```
vtWideString = 15
```

TVarRec type: WideString

29.8.2 Types

```
AnsiChar = Char
```

Alias for 1-byte sized char.

```
Cardinal = LongWord
```

An unsigned 32-bits integer.

```
DWord = LongWord
```

An unsigned 32-bits integer

```
HGLOBAL = Cardinal
```

This is an opaque type.

```
HMODULE = Cardinal
```

This is an opaque type.

```
HRESULT = LongInt
```

32-Bit signed integer.

```
IInterface = IUnknown
```

`IInterface` is the basic interface from which all COM style interfaces descend.

```
Integer = SmallInt
```

The system unit defines `Integer` as a signed 16-bit integer. But when `DELPHI` or `OBJFPC` mode are active, then the objpas unit redefines `Integer` as a 32-bit integer.

```
IntegerArray = Array[0..$efffffff] of Integer
```

Generic array of integer.

```
jmp_buf = packed record
  ebx : LongInt;
  esi : LongInt;
  edi : LongInt;
  bp : Pointer;
  sp : Pointer;
  pc : Pointer;
end
```

Record type to store processor information.

```
MAKEINTRESOURCE = PChar
```

Alias for the PChar (1184) type.

```
PAnsiChar = PChar
```

Alias for PChar (1184) type.

```
PAnsiString = ^AnsiString
```

Pointer to an ansistring type.

```
PBoolean = ^Boolean
```

Pointer to a Boolean type.

```
PByte = ^Byte
```

Pointer to byte (1159) type

```
pcalldesc = ^tcalldesc
```

Pointer to TCallDesc (1191) record.

```
PCardinal = ^Cardinal
```

Pointer to Cardinal (1182) type

```
PChar = ^Char
```

Or the same as a pointer to an array of char. See the reference manual for more information about this type.

```
PClass = ^TClass
```

Pointer to TClass (1191)

```
PCurrency = ^Currency
```

Pointer to currency type.

```
PDate = ^TDateTime
```

Pointer to a TDateTime (1191) type.

```
PDateTime = ^TDateTime
```

Pointer to `TDatetime`

```
PDispatch = ^IDispatch
```

Pointer to IDispatch (1335) interface type

```
pdispdesc = ^tdispdesc
```

Pointer to tdispdesc (1191) record

```
PDouble = ^Double
```

Pointer to double-sized float value.

```
PDWord = ^DWord
```

Pointer to DWord (1182) type

```
pdynarrayindex = ^tdynarrayindex
```

Pointer to tdynarrayindex (1191) type.

```
pdynarraytypeinfo = ^tdynarraytypeinfo
```

Pointer to TDynArrayTypeInfo (1191) type.

```
PError = ^TError
```

Pointer to an Error (1233) type.

```
PEventState = pointer
```

Pointer to EventState, which is an opaque type.

```
PExceptObject = ^TExceptObject
```

Pointer to Exception handler procedural type TExceptProc (1192)

```
PExtended = ^Extended
```

Pointer to extended-sized float value.

```
PGuid = ^TGuid
```

Pointer to TGUID (1193) type.

```
PInt64 = ^Int64
```

Pointer to `Int64` type

```
PInteger = ^Integer
```

Pointer to integer (1183) type

```
PIntegerArray = ^IntegerArray
```

Pointer to IntegerArray (1183) type

```
pinterfaceentry = ^tinterfaceentry
```

Pointer to tinterfaceentry (1193) record.

```
pinterfacetable = ^tinterfacetable
```

Pointer to tinterfacetable (1194) record.

```
PJmp_buf = ^jmp_buf
```

Pointer to jmp_buf (1183) record

```
PLongBool = ^LongBool
```

Pointer to a `LongBool` type.

```
PLongint = ^LongInt
```

Pointer to Longint (1159) type

```
PLongWord = ^LongWord
```

Pointer to `LongWord` type

```
PMemoryManager = ^TMemoryManager
```

Pointer to TMemoryManager (1194) record

```
PMsgStrTable = ^TMsgStrTable
```

Pointer to array of TMsgStrTable (1195) records.

```
PointerArray = Array[0..512*1024*1024-2] of Pointer
```

Generic pointer array.

```
POleVariant = ^OleVariant
```

Pointer to `OleVariant` type.

```
PPAnsiChar = PPChar
```

Alias for PPChar (1186) type.

```
PPChar = ^PChar
```

Pointer to an array of pointers to null-terminated strings.

```
PPCharArray = ^TPCharArray
```

Pointer to TPCharArray (1195) type.

```
PPDispatch = ^PDispatch
```

Pointer to PDispatch (1184) pointer type

```
PPointer = ^Pointer
```

Pointer to a pointer type.

```
PPointerArray = ^PointerArray
```

Pointer to PointerArray (1186) type

```
PPPointer = ^PPointer
```

Pointer to a PPointer (1186) type.

```
PPtrInt = ^PtrInt
```

Pointer to PtrInt (1188) type.

```
PPtrUInt = ^PtrUInt
```

Pointer to unsigned integer of pointer size

```
PPUnknown = ^PUnknown
```

Pointer to untyped pointer

```
PPWideChar = ^PWideChar
```

Pointer to link id="PWideChar"> type.

```
PQWord = ^QWord
```

Pointer to `QWord` type

```
PRTLCriticalSection = ^TRTLCriticalSection
```

Pointer to #rtl.system.TRTLCriticalSection (1195) type.

```
PRTLEvent = pointer
```

Pointer to RTLEvent, which is an opaque type.

```
PShortInt = ^ShortInt
```

Pointer to shortint (1159) type

```
PShortString = ^ShortString
```

Pointer to a shortstring type.

```
PSingle = ^Single
```

Pointer to single-sized float value.

```
PSizeInt = ^SizeInt
```

Pointer to a SizeInt (1189) type

```
PSmallInt = ^SmallInt
```

Pointer to smallint (1159) type

```
pstringmessagetable = ^TStringMessageTable
```

Pointer to TStringMessageTable (1197) record.

```
PText = ^Text
```

Pointer to text file.

```
PtrInt = LongInt
```

Ptrint is a signed integer type which has always the same size as a pointer. Ptrint is considered harmfull and should almost never be used in actual code, because pointers are normally unsigned. For example, consider the following code:

```
getmem(p,2048);                  {Assume the address of p becomes $7ffffff0.}
q:=pointer(ptrint(p)+1024));     {Overflow error.}
writeln(q>p);                    {Incorrect answer.}
```

Ptrint might have a valid use when two pointers are substracted from each other if it is unknown which pointer has the largest address. However, even in this case ptrint causes trouble in case the distance is larger than high(ptrint) and must be used with great care.

The introduction of the ptrint type was a mistake. Please use ptruint (1188) instead.

```
PtrUInt = DWord
```

PtrUInt is an unsigned integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular Cardinal type.

```
PUCS2Char = PWideChar
```

Pointer to UCS2Char (1202) character.

```
PUCS4Char = ^UCS4Char
```

Pointer to UCS4Char (1202)

```
PUCS4CharArray = ^TUCS4CharArray
```

Pointer to array of UCS4Char (1202) characters.

```
PUnknown = ^IUnknown
```

Untyped pointer

```
PUTF8String = ^UTF8String
```

Pointer to UTF8String (1202)

```
pvararray = ^tvararray
```

Pointer to TVarArray (1199) type.

```
pvararraybound = ^tvararraybound
```

Pointer to tvararraybound (1199) type.

```
pvararrayboundarray = ^tvararrayboundarray
```

Pointer to tvararrayboundarray (1199) type.

```
pvararraycoorarray = ^tvararraycoorarray
```

Pointer to tvararraycoorarray (1199) type.

```
pvardata = ^tvardata
```

Pointer to TVarData (1200) record.

```
PVariant = ^Variant
```

Pointer to `Variant` type.

```
pvariantmanager = ^tvariantmanager
```

Pointer to TVariantManager (1201) record.

```
PVarRec = ^TVarRec
```

Pointer to TVarRec (1201) type.

```
PWideChar = ^WideChar
```

Pointer to WChar (1203).

```
PWideString = ^WideString
```

Pointer to widestring type

```
PWord = ^Word
```

Pointer to word (1159) type

```
PWordBool = ^WordBool
```

Pointer to a `WordBool` type.

```
Real = Double
```

Alias for real type

```
real48 = Array[0..5] of Byte
```

TP compatible real type (6 bytes) definition

```
SizeInt = LongInt
```

Signed integer type which fits for sizes

```
SizeUInt = DWord
```

Unsigned Integer type which fits for sizes

```
TAbstractErrorProc = procedure
```

Abstract error handler procedural type.

```
TAllocateThreadVarsHandler = procedure
```

Threadvar allocation callback type for TThreadManager (1198).

```
TAnsiChar = Char
```

Alias for 1-byte sized char.

```
TAssertErrorProc = procedure(const msg: ShortString;
                             const fname: ShortString;lineno: LongInt;
                             erroraddr: pointer)
```

Assert error handler procedural type.

```
TBackTraceStrFunc = function(Addr: Pointer) : ShortString
```

Type for formatting of backtrace dump.

```
TBasicEventCreateHandler = function(EventAttributes: Pointer;
                                    AManualReset: Boolean;
                                    InitialState: Boolean;
                                    const Name: ansistring)
                                     : PEventState
```

callback type for creating eventstate in TThreadManager (1198).

```
TBasicEventHandler = procedure(state: PEventState)
```

Generic callback type for handling eventstate in TThreadManager (1198).

```
TBasicEventWaitForHandler = function(timeout: Cardinal;
                                     state: PEventState) : LongInt
```

Wait for basic event callback type for TThreadManager (1198).

```
TBeginThreadHandler = function(sa: Pointer;stacksize: PtrUInt;
                               ThreadFunction: TThreadFunc;p: pointer;
                               creationFlags: DWord;
                               var ThreadId: TThreadID) : TThreadID
```

Callback for thread start in TThreadManager (1198).

```
TBoundArray = Array of SizeInt
```

Dynamic array of integer.

```
tcalldesc = packed record
  calltype : Byte;
  argcount : Byte;
  namedargcount : Byte;
  argtypes : Array[0..255] of Byte;
end
```

tcalldesc is used to encode the arguments to a dispatch call to an OLE dual interface. It is used on windows only. It describes the arguments to a call.

```
TClass = Class of TObject
```

Class of TObject (1340).

```
TCriticalSectionHandler = procedure(var cs)
```

Generic callback type for critical section handling in TThreadManager (1198).

```
TCtrlBreakHandler = function(CtrlBreak: Boolean) : Boolean
```

TCtrlBreakHandler is the prototype for the CTRL-C handler. If CtrlBreak is True then Ctrl-Break was hit, otherwise CTRL-C was hit. The handlers should return True to signal that the key-combination was handled. If False is returned, then default handling will be used, which by default means an exception will be raised if the sysutils unit is used.

```
TDateTime = Double
```

Encoded Date-Time type.

```
tdispdesc = packed record
  dispid : LongInt;
  restype : Byte;
  calldesc : tcalldesc;
end
```

tcalldesc is used to encode a dispatch call to an OLE dispatch interface. It is used on windows only. It describes the dispath call call.

```
tdynarrayindex = SizeInt
```

A variable of type tdynarrayindex will always have the correct size, suitable for serving as an index in a dynamic array.

```
tdynarraytypeinfo = packed record
  kind : Byte;
  namelen : Byte;
  elesize : SizeInt;
  eletype : pdynarraytypeinfo;
  vartype : LongInt;
end
```

tdynarraytypeinfo describes the structure of a multi-dimensional dynamical array. It is used in the DynArraySetLength (1230) call.

```
TEndThreadHandler = procedure(ExitCode: DWord)
```

Callback for thread end in TThreadManager (1198).

```
TEntryInformation = record
  InitFinalTable : Pointer;
  ThreadvarTablesTable : Pointer;
  asm_exit : procedure;
  PascalMain : procedure;
  valgrind_used : Boolean;
end
```

TEntryInformation is used to initialize a Free Pascal program or library. Under normal circumstances, there should be no need to use this structure directly: it is used by the system unit and special linking units.

```
TError = LongInt
```

Error type, used in variants.

```
TErrorProc = procedure(ErrNo: LongInt;Address: Pointer;Frame: Pointer)
```

Standard error handler procedural type.

```
TExceptObject = record
  FObject : TObject;
  Addr : pointer;
  Next : PExceptObject;
  refcount : LongInt;
  Framecount : LongInt;
  Frames : PPointer;
end
```

TExceptObject is the exception description record which is found on the exception stack.

```
TExceptProc = procedure(Obj: TObject;Addr: Pointer;FrameCount: LongInt;
                        Frame: PPointer)
```

Exception handler procedural type

```
TextFile = Text
```

Alias for Text file type.

```
TFPCHeapStatus = record
  MaxHeapSize : PtrInt;
  MaxHeapUsed : PtrInt;
  CurrHeapSize : PtrInt;
  CurrHeapUsed : PtrInt;
  CurrHeapFree : PtrInt;
end
```

TFPCHeapStatus describes the state of the FPC heap manager. This is not equivalent to the THeapStatus (1193) record defined by Delphi, which contains information not meaningful for the FPC heap manager. The heap status can be retrieved by the GetFPCHeapStatus (1244) call.

```
TGetCurrentThreadIdHandler = function : TThreadID
```

Callback type for retrieving thread ID in TThreadManager (1198).

```
TGuid = packed record
end
```

Standard GUID representation type.

```
THandle = LongInt
```

This type should be considered opaque. It is used to describe file and other handles.

```
THeapStatus = record
  TotalAddrSpace : Cardinal;
  TotalUncommitted : Cardinal;
  TotalCommitted : Cardinal;
  TotalAllocated : Cardinal;
  TotalFree : Cardinal;
  FreeSmall : Cardinal;
  FreeBig : Cardinal;
  Unused : Cardinal;
  Overhead : Cardinal;
  HeapErrorCode : Cardinal;
end
```

`THeapStatus` is the record describing the current heap status. It is returned by the GetHeapStatus (1244) call.

```
TInitThreadVarHandler = procedure(var offset: DWord;size: DWord)
```

Threadvar initialization callback type for TThreadManager (1198).

```
TInterfacedClass = Class of TInterfacedObject
```

`TInterfacedClass` is a descendent of

```
tinterfaceentry = record
  IID : PGuid;
  VTable : Pointer;
  IOffset : PtrInt;
  IIDStr : PShortString;
end
```

`tinterfaceentry` is used to store the list of Interfaces of a class. This list is stored as an array of `tinterfaceentry` records.

```
tinterfaceentrytype = (etStandard,etVirtualMethodResult,
                       etStaticMethodResult,etFieldValue)
```

Table 29.7: Enumeration values for type tinterfaceentrytype

Value	Explanation
etFieldValue	Field value
etStandard	Standard entry
etStaticMethodResult	Statis method
etVirtualMethodResult	Virtual method

This is an internal type for the compiler to encode calls to dispatch interfaces.

```
tinterfacetable = record
  EntryCount : PtrInt;
  Entries : Array[0..0] of tinterfaceentry;
end
```

Record to store list of interfaces of a class.

```
TMemoryManager = record
  NeedLock : Boolean;
  Getmem : function(Size: PtrInt) : Pointer;
  Freemem : function(p: pointer) : PtrInt;
  FrememSize : function(p: pointer;Size: PtrInt) : PtrInt;
  AllocMem : function(Size: PtrInt) : Pointer;
  ReAllocMem : function(var p: pointer;Size: PtrInt) : Pointer;
  MemSize : function(p: pointer) : PtrInt;
  GetHeapStatus : function : THeapStatus;
  GetFPCHeapStatus : function : TFPCHeapStatus;
end
```

`TMemoryManager` describes the memory manager. For more information about the memory manager, see the programmer's reference.

```
TMemoryMutexManager = record
  MutexInit : procedure;
  MutexDone : procedure;
  MutexLock : procedure;
  MutexUnlock : procedure;
end
```

When the heapmanager needs a lock, then the mutex manager is used to handle the lock.

```
TMethod = record
  Code : Pointer;
  Data : Pointer;
end
```

`TMethod` describes a general method pointer, and is used in Run-Time Type Information handling.

```
TMsgStrTable = record
  name : PShortString;
  method : pointer;
end
```

Record used in string message handler table.

```
TPCharArray = packed Array[0..(MaxLongintdivSizeOf(PChar))-1] of PChar
```

Array of PChar

```
TProcedure = procedure
```

Simple procedural type.

```
TReleaseThreadVarsHandler = procedure
```

Threadvar release callback type for TThreadManager (1198).

```
TRelocateThreadVarHandler = function(offset: DWord) : pointer
```

Threadvar relocalization callback type for TThreadManager (1198).

```
TResourceHandle = Cardinal
```

This is an opaque type.

```
TRTLCreateEventHandler = function : PRTLEvent
```

Callback type for creating a `TRTLEvent` type in TThreadManager (1198).

```
TRTLCriticalSection = Opaque type
```

`TRTLCriticalSection` represents a critical section (a mutex). This is an opaque type, it can differ from operating system to operating system. No assumptions should be made about it's structure or contents.

```
TRTLEventHandler = procedure(AEvent: PRTLEvent)
```

Generic TRTLEvent handling type for TThreadManager (1198).

```
TRTLEventHandlerTimeout = procedure(AEvent: PRTLEvent;timeout: LongInt)
```

TRTLEvent timeout handling type for TThreadManager (1198).

```
TRTLEventSyncHandler = procedure(m: trtlmethod;p: TProcedure)
```

Callback type for event sycnhronization in TThreadManager (1198).

```
trtlmethod = procedure of object
```

Callback type for synchronization event.

```
TRuntimeError = (reNone,reOutOfMemory,reInvalidPtr,reDivByZero,
                 reRangeError,reIntOverflow,reInvalidOp,reZeroDivide,
                 reOverflow,reUnderflow,reInvalidCast,reAccessViolation,
                 rePrivInstruction,reControlBreak,reStackOverflow,
                 reVarTypeCast,reVarInvalidOp,reVarDispatch,
                 reVarArrayCreate,reVarNotArray,reVarArrayBounds,
                 reAssertionFailed,reExternalException,reIntfCastError,
                 reSafeCallError,reQuit,reCodesetConversion)
```

Table 29.8: Enumeration values for type TRuntimeError

Value	Explanation
reAccessViolation	Access Violation
reAssertionFailed	Assertion failed error
reCodesetConversion	Code set conversion error
reControlBreak	User pressed CTRL-C
reDivByZero	Division by zero error
reExternalException	An external exception occurred
reIntfCastError	Interface typecast error
reIntOverflow	Integer overflow error
reInvalidCast	Invalid (class) typecast error
reInvalidOp	Invalid operation error
reInvalidPtr	Invalid pointer error
reNone	No error
reOutOfMemory	Out of memory error
reOverflow	Overflow error
rePrivInstruction	Privileged instruction error
reQuit	Quit signal error
reRangeError	Range check error
reSafeCallError	Safecall (IDispInterface) error
reStackOverflow	Stack overflow error
reUnderflow	Underflow error
reVarArrayBounds	Variant array bounds error
reVarArrayCreate	Variant array creation error
reVarDispatch	Variant Dispatch error.
reVarInvalidOp	Invalid variant operation error
reVarNotArray	Variant is not an array error.
reVarTypeCast	Invalid typecase from variant
reZeroDivide	Division by zero error

`TRuntimeError` is used in the Error (1233) procedure to indicate what kind of error should be reported.

```
TSafeCallErrorProc = procedure(error: HRESULT;addr: pointer)
```

Prototype of a safecall error handler routine. `Error` is the error number (passed by the Windows operating system) and `Addr` is the address where the error occurred.

```
TSemaphoreDestroyHandler = procedure(const sem: Pointer)
```

TSemaphoreDestroyHandler is the function prototype to destroy an existing semaphore, as returned by (ThreadManager.SemaphoreInit). It is used by the thread manager (ThreadManager.SemaphoreDest

```
TSemaphorePostHandler = procedure(const sem: Pointer)
```

TSemaphorePostHandler is the function prototype to post an event to the semaphore. It should handle a pointer as returned by the ThreadManager.SemaphoreInit procedure. it's used by the thread manager ThreadManager.SemaphorePost.

```
TSemaphoreWaitHandler = procedure(const sem: Pointer)
```

TSemaphoreWaitHandler is the function prototype to wait on an event on the semaphore (which should be posted to the semaphore with ThreadManager.SemaphorePost). It should handle a pointer as eturned by the ThreadManager.SemaphoreInit procedure. it's used by the thread manager ThreadManager.SemaphoreWait.

```
TSempahoreInitHandler = function : Pointer
```

TSempahoreInitHandler is the function prototype for initializing a semaphore. It is used by the thread manager (ThreadManager.SemaphoreInit) to create semaphores. The function should return a pointer, usable by the other semaphore functions of the thread manager.

```
TStringMessageTable = record
  count : PtrInt;
  msgstrtable : Array[0..0] of TMsgStrTable;
end
```

Record used to describe the string messages handled by a class. It consists of a count, followed by an array of TMsgStrTable (1195) records.

```
TTextLineBreakStyle = (tlbsLF,tlbsCRLF,tlbsCR)
```

Table 29.9: Enumeration values for type TTextLineBreakStyle

Value	Explanation
tlbsCR	Carriage-return (#13, Mac-OS style)
tlbsCRLF	Carriage-return, line-feed (#13#30, Windows style)
tlbsLF	Line-feed only (#10, unix style)

Text line break style. (end of line character)

```
TThreadFunc = function(parameter: pointer) : PtrInt
```

Thread function prototype

```
TThreadGetPriorityHandler = function(threadHandle: TThreadID) : LongInt
```

Callback type for thread priority getting in TThreadManager (1198).

```
TThreadHandler = function(threadHandle: TThreadID) : DWord
```

Generic thread handler callback for TThreadManager (1198).

```
TThreadID = PtrUInt
```

This is an opaque type, it can differ from operating system to operating system.

```
TThreadManager = record
  InitManager : function : Boolean;
  DoneManager : function : Boolean;
  BeginThread : TBeginThreadHandler;
  EndThread : TEndThreadHandler;
  SuspendThread : TThreadHandler;
  ResumeThread : TThreadHandler;
  KillThread : TThreadHandler;
  ThreadSwitch : TThreadSwitchHandler;
  WaitForThreadTerminate : TWaitForThreadTerminateHandler;
  ThreadSetPriority : TThreadSetPriorityHandler;
  ThreadGetPriority : TThreadGetPriorityHandler;
  GetCurrentThreadId : TGetCurrentThreadIdHandler;
  InitCriticalSection : TCriticalSectionHandler;
  DoneCriticalSection : TCriticalSectionHandler;
  EnterCriticalSection : TCriticalSectionHandler;
  LeaveCriticalSection : TCriticalSectionHandler;
  InitThreadVar : TInitThreadVarHandler;
  RelocateThreadVar : TRelocateThreadVarHandler;
  AllocateThreadVars : TAllocateThreadVarsHandler;
  ReleaseThreadVars : TReleaseThreadVarsHandler;
  BasicEventCreate : TBasicEventCreateHandler;
  BasicEventDestroy : TBasicEventHandler;
  BasicEventResetEvent : TBasicEventHandler;
  BasicEventSetEvent : TBasicEventHandler;
  BasiceventWaitFOr : TBasicEventWaitForHandler;
  RTLEventCreate : TRTLCreateEventHandler;
  RTLEventDestroy : TRTLEventHandler;
  RTLEventSetEvent : TRTLEventHandler;
  RTLEventResetEvent : TRTLEventHandler;
  RTLEventWaitFor : TRTLEventHandler;
  RTLEventSync : TRTLEventSyncHandler;
  RTLEventWaitForTimeout : TRTLEventHandlerTimeout;
  SemaphoreInit : TSempahoreInitHandler;
  SemaphoreDestroy : TSemaphoreDestroyHandler;
  SemaphorePost : TSemaphorePostHandler;
  SemaphoreWait : TSemaphoreWaitHandler;
end
```

`TThreadManager` is a record that contains all callbacks needed for the thread handling routines of the Free Pascal Run-Time Library. The thread manager can be set by the SetThreadManager (1314) procedure, and the current thread manager can be retrieved with the GetThreadManager (1245) procedure.

The Windows RTL will set the thread manager automatically to a system thread manager, based on the Windows threading routines. Unix operating systems provide a unit cthreads which implements

threads based on the C library POSIX thread routines. It is not included by default, because it would make the system unit dependent on the C library.

For more information about thread programming, see the programmer's guide.

```
TThreadSetPriorityHandler = function(threadHandle: TThreadID;
                                     Prio: LongInt) : Boolean
```

Callback type for thread priority setting in TThreadManager (1198).

```
TThreadSwitchHandler = procedure
```

Callback type for thread switch in TThreadManager (1198).

```
TUCS4CharArray = Array[0..$efffffff] of UCS4Char
```

Array of UCS4Char (1202) characters.

```
tvararray = record
  dimcount : Word;
  flags : Word;
  elementsize : LongInt;
  lockcount : LongInt;
  data : pointer;
  bounds : tvararrayboundarray;
end
```

`tvararray` is a record describing a variant array. It contains some general data, followed by a number of TVarArrayBound (1199) records equal to the number of dimensions in the array (`dimcoun`).

```
tvararraybound = record
  elementcount : LongInt;
  lowbound : LongInt;
end
```

`tvararraybound` is used to describe one dimension in a variant array.

```
tvararrayboundarray = Array[0..0] of tvararraybound
```

array of tvararraybound (1199) records.

```
tvararraycoorarray = Array[0..0] of LongInt
```

Array of variant array coordinates

```
tvardata = packed record
  vtype : tvartype;
end
```

TVarData is a record representation of a variant. It contains the internal structure of a variant and is handled by the various variant handling routines.

```
tvariantmanager = record
  vartoint : function(const v: variant) : LongInt;
  vartoint64 : function(const v: variant) : Int64;
  vartoword64 : function(const v: variant) : qword;
  vartobool : function(const v: variant) : Boolean;
  vartoreal : function(const v: variant) : extended;
  vartotdatetime : function(const v: variant) : TDateTime;
  vartocurr : function(const v: variant) : currency;
  vartopstr : procedure(var s;const v: variant);
  vartolstr : procedure(var s: ansistring;const v: variant);
  vartowstr : procedure(var s: widestring;const v: variant);
  vartointf : procedure(var intf: IInterface;const v: variant);
  vartodisp : procedure(var disp: IDispatch;const v: variant);
  vartodynarray : procedure(var dynarr: pointer;const v: variant;typeinfo: pointer);
  varfrombool : procedure(var dest: variant;const source: Boolean);
  varfromint : procedure(var dest: variant;const source: LongInt;const Range: LongInt
  varfromint64 : procedure(var dest: variant;const source: Int64);
  varfromword64 : procedure(var dest: variant;const source: qword);
  varfromreal : procedure(var dest: variant;const source: extended);
  varfromtdatetime : procedure(var dest: Variant;const source: TDateTime);
  varfromcurr : procedure(var dest: Variant;const source: Currency);
  varfrompstr : procedure(var dest: variant;const source: ShortString);
  varfromlstr : procedure(var dest: variant;const source: ansistring);
  varfromwstr : procedure(var dest: variant;const source: WideString);
  varfromintf : procedure(var dest: variant;const source: IInterface);
  varfromdisp : procedure(var dest: variant;const source: IDispatch);
  varfromdynarray : procedure(var dest: variant;const source: pointer;typeinfo: point
  olevarfrompstr : procedure(var dest: olevariant;const source: shortstring);
  olevarfromlstr : procedure(var dest: olevariant;const source: ansistring);
  olevarfromvar : procedure(var dest: olevariant;const source: variant);
  olevarfromint : procedure(var dest: olevariant;const source: LongInt;
            const range: ShortInt);
  varop : procedure(var left: variant;const right: variant;opcode: tvarop);
  cmpop : function(const left: variant;const right: variant;const opcode: tvarop)
            : Boolean;
  varneg : procedure(var v: variant);
  varnot : procedure(var v: variant);
  varinit : procedure(var v: variant);
  varclear : procedure(var v: variant);
  varaddref : procedure(var v: variant);
  varcopy : procedure(var dest: variant;const source: variant);
  varcast : procedure(var dest: variant;const source: variant;vartype: LongInt);
  varcastole : procedure(var dest: variant;const source: variant;vartype: LongInt);
  dispinvoke : procedure(dest: pvardata;const source: tvardata;calldesc: pcalldesc;
            params: pointer);
  vararrayredim : procedure(var a: variant;highbound: SizeInt);
  vararrayget : function(const a: variant;indexcount: SizeInt;indices: PSizeInt)
            : variant;
  vararrayput : procedure(var a: variant;const value: variant;indexcount: SizeInt;
            indices: PSizeInt);
  writevariant : function(var t: text;const v: variant;width: LongInt) : Pointer;
```

```
  write0Variant : function(var t: text;const v: Variant) : Pointer;
end
```

`TVariantManager` describes the variant manager as expected by the SetVariantManager (1314) call.

```
tvarop = (opadd,opsubtract,opmultiply,opdivide,opintdivide,opmodulus,
          opshiftleft,opshiftright,opand,opor,opxor,opcompare,opnegate,
          opnot,opcmpeq,opcmpne,opcmplt,opcmple,opcmpgt,opcmpge,oppower)
```

Table 29.10: Enumeration values for type tvarop

Value	Explanation
opadd	Variant operation: Addition.
opand	Variant operation: Binary AND operation
opcmpeq	Variant operation: Compare equal.
opcmpge	Variant operation: Compare larger than or equal
opcmpgt	Variant operation: Compare larger than
opcmple	Variant operation: Compare less than or equal to
opcmplt	Variant operation: Compare less than.
opcmpne	Variant operation: Compare not equal
opcompare	Variant operation: Compare
opdivide	Variant operation: division
opintdivide	Variant operation: integer divide
opmodulus	Variant operation: Modulus
opmultiply	Variant operation: multiplication
opnegate	Variant operation: negation.
opnot	Variant operation: Binary NOT operation.
opor	Variant operation: Binary OR operation
oppower	Variant operation: Power
opshiftleft	Variant operation: Shift left
opshiftright	Variant operation: Shift right
opsubtract	Variant operation: Substraction
opxor	Variant operation: binary XOR operation.

`tvarop` describes a variant operation. It is mainly used for the variant manager to implement the various conversions and mathematical operations on a variant.

```
TVarRec = record
end
```

`TVarRec` is a record generated by the compiler for each element in a `array of const` call. The procedure that receives the constant array receives an array of `TVarRec` elements, with lower bound zero and high bound equal to the number of elements in the array minus one (as returned by `High(Args)`)

```
tvartype = Word
```

Type with size of variant type.

```
TWaitForThreadTerminateHandler = function(threadHandle: TThreadID;
                                          TimeoutMs: LongInt) : DWord
```

Callback type for thread termination in TThreadManager (1198).

```
TWideStringManager = record
  Wide2AnsiMoveProc : procedure(source: PWideChar;var dest: ansistring;len: SizeInt)
  Ansi2WideMoveProc : procedure(source: PChar;var dest: widestring;len: SizeInt);
  UpperWideStringProc : function(const S: WideString) : WideString;
  LowerWideStringProc : function(const S: WideString) : WideString;
  CompareWideStringProc : function(const s1: WideString;const s2: WideString) : PtrI
  CompareTextWideStringProc : function(const s1: WideString;const s2: WideString) : I
  CharLengthPCharProc : function(const Str: PChar) : PtrInt;
  UpperAnsiStringProc : function(const s: ansistring) : ansistring;
  LowerAnsiStringProc : function(const s: ansistring) : ansistring;
  CompareStrAnsiStringProc : function(const S1: ansistring;const S2: ansistring) : P
  CompareTextAnsiStringProc : function(const S1: ansistring;const S2: ansistring) : I
  StrCompAnsiStringProc : function(S1: PChar;S2: PChar) : PtrInt;
  StrICompAnsiStringProc : function(S1: PChar;S2: PChar) : PtrInt;
  StrLCompAnsiStringProc : function(S1: PChar;S2: PChar;MaxLen: PtrUInt) : PtrInt;
  StrLICompAnsiStringProc : function(S1: PChar;S2: PChar;MaxLen: PtrUInt) : PtrInt;
  StrLowerAnsiStringProc : function(Str: PChar) : PChar;
  StrUpperAnsiStringProc : function(Str: PChar) : PChar;
  ThreadInitProc : procedure;
  ThreadFiniProc : procedure;
end
```

`TWideStringManager` contains the definition of the widestring manager.

```
UCS2Char = WideChar
```

UCS2 unicode character.

```
UCS4Char =
```

UCS unicode character (unsigned 32 bit word)

```
UCS4String = Array of UCS4Char
```

String of UCS4Char (1202) characters.

```
UInt64 = QWord
```

Unsigned 64-bit integer

```
UTF8String = ansistring
```

UTF-8 unicode (ansi) string.

```
ValSInt = LongInt
```

Integer with the same size as the return code of the Val (1331) function.

```
ValUInt = Cardinal
```

Integer with the same size as the return code of the Val (1331) function.

```
WChar = Widechar
```

Wide char (16-bit sized char)

29.8.3 Variables

```
argc : LongInt;external name operatingsystem_parameter_argc
```

`argc` contains the number of command-line arguments passed to the program by the OS. It is not available on all systems.

```
argv : PPChar;external name operatingsystem_parameter_argv
```

`argv` contains a pointer to a `nil`-terminated array of null-terminated strings, containing the command-line arguments passed to the program by the OS. It is not available on all systems.

```
cmdline : PChar = nil
```

Current command-line.

```
DispCallByIDProc : pointer
```

`VarDispProc` is called by the compiler if it needs to perform an interface call from a variant which contains a dispatch interface. For instance, the following call:

```
Var
  V : OleVariant;
begin
  (V as IWord).OpenDocument('c:\temp\mydoc.doc');
end;
```

where `IWord` is a dispatch interface is encoded by the compiler and passed to `DispCallByIDProc`. This pointer must be set by a routine that calls the OS COM handling routines.

```
envp : PPChar;external name operatingsystem_parameter_envp
```

`envp` contains a pointer to a `nil`-terminated array of null-terminated strings, containing the environment variables passed to the program by the OS. It is not available on all systems.

```
ErrOutput : Text
```

`ErrOutput` is provided for Delphi compatibility.

```
ExitCode : LongInt;public name operatingsystem_result
```

Exit code for the program, will be communicated to the OS on exit.

```
fpc_threadvar_relocate_proc : pointer;public name FPC_THREADVAR_RELOCATE
```

```
InOutRes : Word
```

Result of last I/O operation. Read-Only.

```
Input : Text
```

Standard input text file.

```
IsConsole : Boolean
```

`True` for console applications, `False` for GUI applications.

```
IsLibrary : Boolean = false
```

`True` if the current module is a library. Otherwise module is an excutable

```
Output : Text
```

Standard output text file.

```
RandSeed : Cardinal
```

Seed for Random (1297) function.

```
ReturnNilIfGrowHeapFails : Boolean
```

`ReturnNilIfGrowHeapFails` describes what happens if there is no more memory available from the operating system. if set to `True` the memory manager will return `Nil`. If set to `False` then a run-time error will occur.

```
softfloat_exception_flags : Byte
```

Current soft float exception flags

```
softfloat_exception_mask : Byte
```

Current soft float exception mask

```
StackBottom : Pointer
```

Current stack bottom.

```
StackLength : SizeUInt
```

Maximum stack length.

`StackTop : Pointer`

`StackTop` contains the top of the stack for the current process. It is used to check the heap on some operating systems, and is set by the system unit initialization code. Do not use or modify this value.

`StdErr : Text`

Standard diagnostic output text file.

`StdOut : Text`

Alias for Output (1204).

`ThreadID : TThreadID`

Current Thread ID.

`widestringmanager : TWideStringManager`

Contains the current widestring manager. Do not use directly.

29.9 Procedures and functions

29.9.1 abs

Synopsis: Calculate absolute value

Declaration:
```
function abs(l: LongInt) : LongInt
function abs(l: Int64) : Int64
function abs(d: ValReal) : ValReal
```

Visibility: default

Description: `Abs` returns the absolute value of a variable. The result of the function has the same type as its argument, which can be any numerical type.

Errors: None.

See also: Round (1305)

Listing: ./refex/ex1.pp

```
Program Example1;

{ Program to demonstrate the Abs function. }

Var
  r : real;
  i : integer;

begin
  r:=abs(-1.0);    { r:=1.0 }
  i:=abs(-21);     { i:=21 }
end.
```

29.9.2 AbstractError

Synopsis: Generate an abstract error.

Declaration: `procedure AbstractError`

Visibility: default

Description: `AbstractError` generates an abstract error (run-time error 211). If the AbstractErrorProc (1161) constant is set, it will be called instead.

Errors: This routine causes a run-time error 211.

See also: AbstractErrorProc (1161)

29.9.3 AcquireExceptionObject

Synopsis: Obtain a reference to the current exception object

Declaration: `function AcquireExceptionObject : Pointer`

Visibility: default

Description: `AcquireExceptionObject` returns the current exception object. It raises the reference count of the exception object, so it will not be freed. Calling this method is only valid within an except block.

The effect of this function is countered by re-raising an exception via `raise;`

To make sure that the exception object is released when it is no longer needed, ReleaseExceptionObject (1301) must be called when the reference is no longer needed.

Errors: If there is no current exception, a run-time error 231 will occur.

See also: ReleaseExceptionObject (1301)

29.9.4 AddExitProc

Synopsis: Add an exit procedure to the exit procedure chain.

Declaration: `procedure AddExitProc(Proc: TProcedure)`

Visibility: default

Description: `AddExitProc` adds `Proc` to the exit procedure chain. At program exit, all procedures added in this way will be called in reverse order.

Errors: None.

See also: ExitProc (1164)

29.9.5 Addr

Synopsis: Return address of a variable

Declaration: `function Addr(X: TAnytype) : Pointer`

Visibility: default

Description: `Addr` returns a pointer to its argument, which can be any type, or a function or procedure name. The returned pointer isn't typed. The same result can be obtained by the `@` operator, which can return a typed pointer (\progref).

Errors: None

See also: SizeOf (1315)

Listing: ./refex/ex2.pp

```
Program Example2;

{ Program to demonstrate the Addr function. }

Const Zero : integer = 0;

Var p : pointer;
    i : Integer;

begin
  p:=Addr(p);      { P points to itself }
  p:=Addr(I);      { P points to I }
  p:=Addr(Zero);   { P points to 'Zero' }
end.
```

29.9.6 Align

Synopsis: Return aligned version of an address

Declaration:
```
function Align(Addr: PtrUInt;Alignment: PtrUInt) : PtrUInt
function Align(Addr: Pointer;Alignment: PtrUInt) : Pointer
```

Visibility: default

Description: `Align` returns `Address`, aligned to `Alignment` bytes.

Errors: None.

29.9.7 AllocMem

Synopsis: Allocate and clear memory.

Declaration:
```
function AllocMem(Size: PtrInt) : pointer
```

Visibility: default

Description: `AllocMem` calls getmem GetMem (1244), and clears the allocated memory, i.e. the allocated memory is filled with `Size` zero bytes.

See also: GetMem (1244)

29.9.8 AnsiToUtf8

Synopsis: Convert ansi string to UTF-8 string

Declaration:
```
function AnsiToUtf8(const s: ansistring) : UTF8String
```

Visibility: default

Description: `AnsiToUtf8` converts the ansistring `S` to a WideString in UTF-8 format.

Errors: None.

See also: Utf8toAnsi (1330)

29.9.9 Append

Synopsis: Open a file in append mode

Declaration: `procedure Append(var t: Text)`

Visibility: default

Description: `Append` opens an existing file in append mode. Any data written to `F` will be appended to the file. Only text files can be opened in append mode. After a call to `Append`, the file `F` becomes write-only. File sharing is not taken into account when calling `Append`.

Errors: If the file doesn't exist when appending, a run-time error will be generated. This behaviour has changed on Windows and Linux platforms, where in versions prior to 1.0.6, the file would be created in append mode.

See also: Rewrite (1303), Close (1217), Reset (1302)

Listing: ./refex/ex3.pp

```
Program Example3;

{ Program to demonstrate the Append function. }

Var f : text;

begin
  Assign (f,'test.txt');
  Rewrite (f);            { file is opened for write, and emptied }
  WriteIn (F,'This is the first line of text.txt');
  close (f);
  Append(f);              { file is opened for write, but NOT emptied.
                            any text written to it is appended.}
  WriteIn (f,'This is the second line of text.txt');
  close (f);
end.
```

29.9.10 arctan

Synopsis: Calculate inverse tangent

Declaration: `function arctan(d: ValReal) : ValReal`

Visibility: default

Description: `Arctan` returns the Arctangent of `X`, which can be any Real type. The resulting angle is in radial units.

Errors: None

See also: Sin (1315), Cos (1224)

Listing: ./refex/ex4.pp

```
Program Example4;

{ Program to demonstrate the ArcTan function. }

Var R : Real;

begin
  R:=ArcTan(0);       { R:=0 }
  R:=ArcTan(1)/pi;    { R:=0.25 }
end.
```

29.9.11 ArrayStringToPPchar

Synopsis: Concert an array of string to an array of null-terminated strings

Declaration:
```
function ArrayStringToPPchar(const S: Array of AnsiString;
                             reserveentries: LongInt) : PPChar
```

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: StringToPPChar (1320)

29.9.12 Assert

Synopsis: Check validity of a given condition.

Declaration:
```
procedure Assert(Expr: Boolean)
procedure Assert(Expr: Boolean;const Msg: String)
```

Visibility: default

Description: With assertions on, `Assert` tests if `expr` is false, and if so, aborts the application with a Runtime error 227 and an optional error message in `msg`. If `expr` is true, program execution continues normally. If assertions are not enabled at compile time, this routine does nothing, and no code is generated for the `Assert` call. Enabling and disabling assertions at compile time is done via the `\$C` or `\$ASSERTIONS` compiler switches. These are global switches. The default behavior of the assert call can be changed by setting a new handler in the `AssertErrorProc` variable. Sysutils overrides the default handler to raise a `EAssertionFailed` exception.

Errors: None.

See also: Halt (1247), Runerror (1307)

29.9.13 Assign

Synopsis: Assign a name to a file

Declaration:
```
procedure Assign(var f: File;const Name: String)
procedure Assign(var f: File;p: PChar)
procedure Assign(var f: File;c: Char)
procedure Assign(var f: TypedFile;const Name: String)
procedure Assign(var f: TypedFile;p: PChar)
procedure Assign(var f: TypedFile;c: Char)
procedure Assign(var t: Text;const s: String)
procedure Assign(var t: Text;p: PChar)
procedure Assign(var t: Text;c: Char)
```

Visibility: default

Description: `Assign` assigns a name to `F`, which can be any file type. This call doesn't open the file, it just assigns a name to a file variable, and marks the file as closed.

Errors: None.

See also: Reset (1302), Rewrite (1303), Append (1208)

Listing: ./refex/ex5.pp

```
Program Example5;

{ Program to demonstrate the Assign function. }

Var F : text;

begin
  Assign (F,'');
  Rewrite (f);
  { The following can be put in any file by redirecting it
    from the command line.}
  WriteIn (f,'This goes to standard output !');
  Close (f);
  Assign (F,'Test.txt');
  rewrite (f);
  writeln (f,'This doesn''t go to standard output !');
  close (f);
end.
```

29.9.14 Assigned

Synopsis: Check if a pointer is valid

Declaration: `function Assigned(P: Pointer) : Boolean`

Visibility: default

Description: `Assigned` returns `True` if `P` is non-nil and retuns `False` of `P` is nil. The main use of Assigned is that Procedural variables, method variables and class-type variables also can be passed to `Assigned`.

Errors: None

See also: New (1266)

Listing: ./refex/ex96.pp

```
Program Example96;

{ Program to demonstrate the Assigned function. }

Var P : Pointer;

begin
  If Not Assigned(P) then
    WriteIn ('Pointer is initially NIL');
  P:=@P;
  If Not Assigned(P) then
    WriteIn('Internal inconsistency')
  else
    WriteIn('All is well in FPC')
end.
```

29.9.15 BasicEventCreate

Synopsis: Obsolete. Don't use

Declaration:
```
function BasicEventCreate(EventAttributes: Pointer;
                          AManualReset: Boolean;InitialState: Boolean;
                          const Name: ansistring) : PEventState
```

Visibility: default

Description: `BasicEventCreate` is obsolete, use RTLEventCreate (1305) instead.

See also: RTLEventCreate (1305)

29.9.16 basiceventdestroy

Synopsis: Obsolete. Don't use

Declaration:
```
procedure basiceventdestroy(state: PEventState)
```

Visibility: default

Description: `basiceventdestroy` is obsolete. Use RTLEventDestroy (1306) instead.

See also: RTLEventDestroy (1306)

29.9.17 basiceventResetEvent

Synopsis: Obsolete. Don't use

Declaration:
```
procedure basiceventResetEvent(state: PEventState)
```

Visibility: default

Description: `basiceventResetEvent` is obsolete. Use RTLEventResetEvent (1306) instead.

See also: RTLEventResetEvent (1306)

29.9.18 basiceventSetEvent

Synopsis: Obsolete. Don't use

Declaration:
```
procedure basiceventSetEvent(state: PEventState)
```

Visibility: default

Description: `basiceventSetEvent` is obsolete. Use RTLEventSetEvent (1306) instead.

See also: RTLEventSetEvent (1306)

29.9.19 basiceventWaitFor

Synopsis: Obsolete. Don't use

Declaration:
```
function basiceventWaitFor(Timeout: Cardinal;state: PEventState)
                                  : LongInt
```

Visibility: default

Description: `basiceventwaitfor` is obsolete. Use RTLEventWaitFor (1306) instead.

See also: RTLEventWaitFor (1306)

29.9.20 BeginThread

Synopsis: Start a new thread.

Declaration:
```
function BeginThread(sa: Pointer;stacksize: SizeUInt;
                        ThreadFunction: TThreadFunc;p: pointer;
                        creationFlags: DWord;var ThreadId: TThreadID)
                         : TThreadID
function BeginThread(ThreadFunction: TThreadFunc) : TThreadID
function BeginThread(ThreadFunction: TThreadFunc;p: pointer) : TThreadID
function BeginThread(ThreadFunction: TThreadFunc;p: pointer;
                        var ThreadId: TThreadID) : TThreadID
function BeginThread(ThreadFunction: TThreadFunc;p: pointer;
                        var ThreadId: TThreadID;const stacksize: SizeUInt)
                         : TThreadID
```

Visibility: default

Description: `BeginThread` starts a new thread and executes `ThreadFunction` in the new thread. If `P` is specified, then it is passed to `ThreadFunction`. If `ThreadId` is specified, it is filled with the thread ID of the newly started thread. If `StackSize` is specified, it is set as the stack size for the new thread. If none is specified, a default stack size of 32Kb is used.

The function returns the thread ID on succes, or 0 if an error occurred.

Errors: On error, a nonzero value is returned.

See also: EndThread (1230)

29.9.21 BEtoN

Synopsis: Convert Big Endian-ordered integer to Native-ordered integer

Declaration:
```
function BEtoN(const AValue: SmallInt) : SmallInt
function BEtoN(const AValue: Word) : Word
function BEtoN(const AValue: LongInt) : LongInt
function BEtoN(const AValue: DWord) : DWord
function BEtoN(const AValue: Int64) : Int64
function BEtoN(const AValue: QWord) : QWord
```

Visibility: default

Description: `BEToN` will rearrange the bytes in a Big-Endian number to the native order for the current processor. That is, for a big-endian processor, it will do nothing, and for a little-endian processor, it will invert the order of the bytes.

See also: LEtoN (1261), NtoBE (1266), NtoLE (1267)

29.9.22 binStr

Synopsis: Convert integer to string with binary representation.

Declaration:
```
function binStr(Val: LongInt;cnt: Byte) : shortstring
function binStr(Val: Int64;cnt: Byte) : shortstring
function binStr(Val: qword;cnt: Byte) : shortstring
```

Visibility: default

Description: `BinStr` returns a string with the binary representation of `Value`. The string has at most `cnt` characters. (i.e. only the `cnt` rightmost bits are taken into account) To have a complete representation of any longint-type value, 32 bits are needed, i.e. `cnt=32`

Errors: None.

See also: Str (1318), Val (1331), HexStr (1247), OctStr (1267)

Listing: ./refex/ex82.pp

```
Program example82;

{ Program to demonstrate the BinStr function }

Const Value = 45678;

Var I : longint;

begin
  For I:=8 to 20 do
    WriteIn (BinStr(Value,I):20);
end.
```

29.9.23 BlockRead

Synopsis: Read data from an untyped file into memory

Declaration:
```
procedure BlockRead(var f: File;var Buf;count: Int64;var Result: Int64)
procedure BlockRead(var f: File;var Buf;count: LongInt;
                    var Result: LongInt)
procedure BlockRead(var f: File;var Buf;count: Cardinal;
                    var Result: Cardinal)
procedure BlockRead(var f: File;var Buf;count: Word;var Result: Word)
procedure BlockRead(var f: File;var Buf;count: Word;var Result: Integer)
procedure BlockRead(var f: File;var Buf;count: Int64)
```

Visibility: default

Description: `Blockread` reads `count` or less records from file `F`. A record is a block of bytes with size specified by the Rewrite (1303) or Reset (1302) statement. The result is placed in `Buffer`, which must contain enough room for `Count` records. The function cannot read partial records. If `Result` is specified, it contains the number of records actually read. If `Result` isn't specified, and less than `Count` records were read, a run-time error is generated. This behavior can be controlled by the \var{\{\$i\}} switch.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Blockwrite (1214), Close (1217), Reset (1302), Assign (1210)

Listing: ./refex/ex6.pp

```
Program Example6;

{ Program to demonstrate the BlockRead and BlockWrite functions. }

Var Fin, fout : File;
    NumRead, NumWritten : Word;
    Buf : Array[1..2048] of byte;
    Total : Longint;

begin
  Assign (Fin, Paramstr(1));
  Assign (Fout,Paramstr(2));
  Reset (Fin,1);
  Rewrite (Fout,1);
  Total:=0;
  Repeat
    BlockRead (Fin,buf,Sizeof(buf),NumRead);
    BlockWrite (Fout,Buf,NumRead,NumWritten);
    inc(Total,NumWritten);
  Until (NumRead=0) or (NumWritten<>NumRead);
  Write ('Copied ',Total,' bytes from file ',paramstr(1));
  Writeln (' to file ',paramstr(2));
  close(fin);
  close(fout);
end.
```

29.9.24 BlockWrite

Synopsis: Write data from memory to an untyped file

Declaration:
```
procedure BlockWrite(var f: File;const Buf;Count: Int64;
                     var Result: Int64)
```

```
procedure BlockWrite(var f: File;const Buf;Count: LongInt;
                     var Result: LongInt)
procedure BlockWrite(var f: File;const Buf;Count: Cardinal;
                     var Result: Cardinal)
procedure BlockWrite(var f: File;const Buf;Count: Word;var Result: Word)
procedure BlockWrite(var f: File;const Buf;Count: Word;
                     var Result: Integer)
procedure BlockWrite(var f: File;const Buf;Count: LongInt)
```

Visibility: default

Description: `BlockWrite` writes `count` records from `buffer` to the file `F`.A record is a block of bytes with size specified by the Rewrite (1303) or Reset (1302) statement. If the records couldn't be written to disk, a run-time error is generated. This behavior can be controlled by the \var{\{\$i\}} switch.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Blockread (1213), Close (1217), Rewrite (1303), Assign (1210)

29.9.25 Break

Synopsis: Exit current loop construct.

Declaration: `procedure Break`

Visibility: default

Description: `Break` jumps to the statement following the end of the current repetitive statement. The code between the `Break` call and the end of the repetitive statement is skipped. The condition of the repetitive statement is NOT evaluated.

This can be used with `For`, var{repeat} and `While` statements.

Note that while this is a procedure, `Break` is a reserved word and hence cannot be redefined.

Errors: None.

See also: Continue (1223), Exit (1234)

Listing: ./refex/ex87.pp

```
Program Example87;

{ Program to demonstrate the Break function. }

Var I : longint;

begin
  I:=0;
  While I<10 Do
    begin
    Inc(I);
    If I>5 Then
      Break;
    WriteIn (i);
    end;
  I:=0;
  Repeat
```

```
    Inc(I);
    If I>5 Then
      Break;
    WriteIn (i);
  Until I>=10;
  For I:=1 to 10 do
    begin
    If I>5 Then
      Break;
    WriteIn (i);
    end;
end.
```

29.9.26 chdir

Synopsis: Change current working directory.

Declaration: `procedure chdir(const s: String)`

Visibility: default

Description: `Chdir` changes the working directory of the process to `S`.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Mkdir (1264), Rmdir (1304)

Listing: ./refex/ex7.pp

```
Program Example7;

{ Program to demonstrate the ChDir function. }

begin
  {$I-}
  ChDir (ParamStr(1));
  if IOresult<>0 then
    WriteIn ('Cannot change to directory : ',paramstr (1));
end.
```

29.9.27 chr

Synopsis: Convert byte value to character value

Declaration: `function chr(b: Byte) : Char`

Visibility: default

Description: `Chr` returns the character which has ASCII value `X`.

Historical note:

Originally, Pascal did not have typecasts and chr was a necessary function in order to do certain operations on ASCII values of characters. With the arrival of typecasting a generic approach became possible, making chr mostly obselete. However, chr is not considered deprecated and remains in wide use today.

Errors: None.

See also: Ord (1292), Str (1318)

Listing: ./refex/ex8.pp

```
Program Example8;

{ Program to demonstrate the Chr function. }

begin
  Write (chr(10),chr(13)); { The same effect as WriteIn; }
end.
```

29.9.28 Close

Synopsis: Close a file

Declaration:
```
procedure Close(var f: File)
procedure Close(var t: Text)
```

Visibility: default

Description: `Close` flushes the buffer of the file `F` and closes `F`. After a call to `Close`, data can no longer be read from or written to `F`. To reopen a file closed with `Close`, it isn't necessary to assign the file again. A call to Reset (1302) or Rewrite (1303) is sufficient.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Assign (1210), Reset (1302), Rewrite (1303), Flush (1240)

Listing: ./refex/ex9.pp

```
Program Example9;

{ Program to demonstrate the Close function. }

Var F : text;

begin
 Assign (f,'Test.txt');
 ReWrite (F);
 WriteIn (F,'Some text written to Test.txt');
 close (f); { Flushes contents of buffer to disk,
              closes the file. Omitting this may
              cause data NOT to be written to disk.}
end.
```

29.9.29 CompareByte

Synopsis: Compare 2 memory buffers byte per byte

Declaration:
```
function CompareByte(const buf1;const buf2;len: SizeInt) : SizeInt
```

Visibility: default

Description: `CompareByte` compares two memory regions `buf1`,`buf2` on a byte-per-byte basis for a total of `len` bytes.

The function returns one of the following values:

less than 0if `buf1` and `buf2` contain different bytes in the first `len` bytes, and the first such byte is smaller in `buf1` than the byte at the same position in `buf2`.

0if the first `len` bytes in `buf1` and `buf2` are equal. \item [greater than 0] if `buf1` and `buf2` contain different bytes in the first `len` bytes, and the first such byte is larger in `buf1` than the byte at the same position in `buf2`.

Errors: None.

See also: CompareChar (1218), CompareWord (1221), CompareDWord (1220)

Listing: ./refex/ex99.pp

```
Program Example99;

{ Program to demonstrate the CompareByte function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of byte;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' positions are ');
    if CompareByte(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;


begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=I;
    If I<=HalfArraySize Then
      Buf2[I]:=I
    else
      Buf2[i]:=HalfArraySize-I;
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
end.
```

29.9.30 CompareChar

Synopsis: ompare 2 memory buffers character per character

Declaration: `function CompareChar(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareChar` compares two memory regions `buf1`,`buf2` on a character-per-character basis for a total of `len` characters.

The `CompareChar0` variant compares `len` bytes, or until a zero character is found.

The function returns one of the following values:

-1 if `buf1` and `buf2` contain different characters in the first `len` positions, and the first such character is smaller in `buf1` than the character at the same position in `buf2`.

0 if the first `len` characters in `buf1` and `buf2` are equal.

1 if `buf1` and `buf2` contain different characters in the first `len` positions, and the first such character is larger in `buf1` than the character at the same position in `buf2`.

Errors: None.

See also: CompareByte (1217), CompareWord (1221), CompareDWord (1220)

Listing: ./refex/ex100.pp

```
Program Example100;

{ Program to demonstrate the CompareChar function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of char;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' characters are ');
    if CompareChar(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;

  Procedure CheckNullPos(Len : Longint);

  Begin
    Write('First ',Len,' non-null characters are ');
    if CompareChar0(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;

begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=chr(I);
    If I<=HalfArraySize Then
      Buf2[I]:=chr(I)
```

```
    else
      Buf2[i]:=chr(HalfArraySize-I);
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
  For I:=1 to 4 do
    begin
    buf1[Random(ArraySize)+1]:=Chr(0);
    buf2[Random(ArraySize)+1]:=Chr(0);
    end;
  Randomize;
  CheckNullPos(HalfArraySize div 2);
  CheckNullPos(HalfArraySize);
  CheckNullPos(HalfArraySize+1);
  CheckNullPos(HalfArraySize + HalfArraySize Div 2);
end.
```

29.9.31 CompareChar0

Synopsis: Compare two buffers character by character till a null-character is reached.

Declaration: `function CompareChar0(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareChar0` compares 2 buffers `buf1` and `buf2` for a maximum length of `len` or till a null character is reached in either buffer. The result depends on the contents of the buffers:

< 0 If `buf1` contains a character less than the corresponding character in `buf2`.

0 If both buffers are equal

> 0 If `buf1` contains a character greater than the corresponding character in `buf2`.

Errors: None.

See also: CompareByte (1217), CompareChar (1218), CompareDWord (1220), CompareWord (1221)

29.9.32 CompareDWord

Synopsis: Compare 2 memory buffers DWord per DWord

Declaration: `function CompareDWord(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareDWord` compares two memory regions `buf1`,`buf2` on a DWord-per-DWord basis for a total of `len` DWords. (A DWord is 4 bytes).

The function returns one of the following values:

-1 if `buf1` and `buf2` contain different DWords in the first `len` DWords, and the first such DWord is smaller in `buf1` than the DWord at the same position in `buf2`.

0 if the first `len` DWords in `buf1` and `buf2` are equal.

1 if `buf1` and `buf2` contain different DWords in the first `len` DWords, and the first such DWord is larger in `buf1` than the DWord at the same position in `buf2`.

Errors: None.

See also: CompareChar (1218), CompareByte (1217), CompareWord (1221)

Listing: ./refex/ex101.pp

```
Program Example101;

{ Program to demonstrate the CompareDWord function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of Dword;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' DWords are ');
    if CompareDWord(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;


begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=I;
    If I<=HalfArraySize Then
      Buf2[I]:=I
    else
      Buf2[i]:=HalfArraySize-I;
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
end.
```

29.9.33 CompareWord

Synopsis: Compare 2 memory buffers word per word

Declaration: `function CompareWord(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareWord` compares two memory regions `buf1`,`buf2` on a Word-per-Word basis for a total of `len` Words. (A Word is 2 bytes).

The function returns one of the following values:

-1 if `buf1` and `buf2` contain different Words in the first `len` Words, and the first such Word is smaller in `buf1` than the Word at the same position in `buf2`.

0if the first `len` Words in `buf1` and `buf2` are equal.

1if `buf1` and `buf2` contain different Words in the first `len` Words, and the first such Word is larger in `buf1` than the Word at the same position in `buf2`.

Errors: None.

See also: CompareChar (1218), CompareByte (1217), CompareDWord (1220)

Listing: ./refex/ex102.pp

```
Program Example102;

{ Program to demonstrate the CompareWord function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of Word;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' words are ');
    if CompareWord(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    WriteIn('equal');
  end;


begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=I;
    If I<=HalfArraySize Then
      Buf2[I]:=I
    else
      Buf2[i]:=HalfArraySize-I;
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
end.
```

29.9.34 Concat

Synopsis: Append one string to another.

Declaration:
```
function Concat(const S1: String;const S2: String;const S3: String;
                const Sn: String) : String
```

Visibility: default

Description: `Concat` concatenates the strings `S1`,`S2` etc. to one long string. The resulting string is truncated at a length of 255 bytes. The same operation can be performed with the + operation.

Errors: None.

See also: Copy (1224), Delete (1227), Insert (1255), Pos (1295), Length (1260)

Listing: ./refex/ex10.pp

```
Program Example10;

{ Program to demonstrate the Concat function. }
Var
  S : String;

begin
  S:=Concat('This can be done',' Easier ','with the + operator !');
end.
```

29.9.35 Continue

Synopsis: Continue with next loop cycle.

Declaration: `procedure Continue`

Visibility: default

Description: `Continue` jumps to the end of the current repetitive statement. The code between the `Continue` call and the end of the repetitive statement is skipped. The condition of the repetitive statement is then checked again.

This can be used with `For`, var{repeat} and `While` statements.

Note that while this is a procedure, `Continue` is a reserved word and hence cannot be redefined.

Errors: None.

See also: Break (1215), Exit (1234)

Listing: ./refex/ex86.pp

```
Program Example86;

{ Program to demonstrate the Continue function. }

Var I : longint;

begin
  I:=0;
  While I<10 Do
    begin
    Inc(I);
    If I<5 Then
      Continue;
    WriteIn (i);
    end;
  I:=0;
  Repeat
    Inc(I);
    If I<5 Then
      Continue;
    WriteIn (i);
```

```
  Until I>=10;
  For I:=1 to 10 do
    begin
    If I<5 Then
      Continue;
    WriteLn (i);
    end;
end.
```

29.9.36 Copy

Synopsis: Copy part of a string.

Declaration:
```
function Copy(S: AStringType;Index: Integer;Count: Integer) : String
function Copy(A: DynArrayType;Index: Integer;Count: Integer) : DynArray
```

Visibility: default

Description: `Copy` returns a string which is a copy if the `Count` characters in `S`, starting at position `Index`. If `Count` is larger than the length of the string `S`, the result is truncated. If `Index` is larger than the length of the string `S`, then an empty string is returned. `Index` is 1-based.

For dynamical arrays, Copy returns a new dynamical array of the same type as the original one, and copies `Count` elements from the old array, starting at position `Index`.

Errors: None.

See also: Delete (1227), Insert (1255), Pos (1295)

Listing: ./refex/ex11.pp

```
Program Example11;

{ Program to demonstrate the Copy function. }

Var S,T : String;

begin
  T:='1234567';
  S:=Copy (T,1,2);    { S:='12'   }
  S:=Copy (T,4,2);    { S:='45'   }
  S:=Copy (T,4,8);    { S:='4567' }
end.
```

29.9.37 cos

Synopsis: Calculate cosine of angle

Declaration:
```
function cos(d: ValReal) : ValReal
```

Visibility: default

Description: `Cos` returns the cosine of `X`, where X is an angle, in radians. If the absolute value of the argument is larger than 2$\hat{6}$3, then the result is undefined.

Errors: None.

See also: Arctan (1208), Sin (1315)

Listing: ./refex/ex12.pp

```
Program Example12;

{ Program to demonstrate the Cos function. }

Var R : Real;

begin
  R:=Cos(Pi);      { R:=-1 }
  R:=Cos(Pi/2);    { R:=0  }
  R:=Cos(0);       { R:=1  }
end.
```

29.9.38 Cseg

Synopsis: Return code segment

Declaration: `function Cseg : Word`

Visibility: default

Description: `CSeg` returns the Code segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32 bit compiler.

Errors: None.

See also: DSeg (1228), Seg (1309), Ofs (1268), Ptr (1296)

Listing: ./refex/ex13.pp

```
Program Example13;

{ Program to demonstrate the CSeg function. }

var W : word;

begin
  W:=CSeg; {W:=0, provided for compatibility,
                 FPC is 32 bit.}
end.
```

29.9.39 Dec

Synopsis: Decrease value of variable

Declaration:
```
procedure Dec(var X: TOrdinal)
procedure Dec(var X: TOrdinal;Decrement: TOrdinal)
```

Visibility: default

Description: `Dec` decreases the value of `X` with `Decrement`. If `Decrement` isn't specified, then 1 is taken as a default.

Errors: A range check can occur, or an underflow error, if an attempt it made to decrease `X` below its minimum value.

See also: Inc (1250)

Listing: ./refex/ex14.pp

```
Program Example14;

{ Program to demonstrate the Dec function. }

Var
  I  : Integer;
  L  : Longint;
  W  : Word;
  B  : Byte;
  Si : ShortInt;

begin
 I:=1;
 L:=2;
 W:=3;
 B:=4;
 Si:=5;
 Dec (i);      { i:=0  }
 Dec (L,2);    { L:=0  }
 Dec (W,2);    { W:=1  }
 Dec (B,-2);   { B:=6  }
 Dec (Si,0);   { Si:=5 }
end.
```

29.9.40 DefaultAnsi2WideMove

Synopsis: Standard implementation of Ansi to Widestring conversion routine

Declaration:
```
procedure DefaultAnsi2WideMove(source: PChar;var dest: widestring;
                               len: SizeInt)
```

Visibility: default

Description: `DefaultAnsi2WideMove` simply copies each character of the null-terminated ansi-string `Source` to the corresponding WideChar in `Dest`. At most `Len` characters will be copied.

Errors: None.

See also: DefaultWide2AnsiMove (1226)

29.9.41 DefaultWide2AnsiMove

Synopsis: Standard implementation of Widestring to Ansi conversion routine

Declaration:
```
procedure DefaultWide2AnsiMove(source: PWideChar;var dest: ansistring;
                               len: SizeInt)
```

Visibility: default

Description: `DefaultWide2AnsiMove` simply copies each character from `Source` having an ordinal value of less than 255 to the corresponding character in `Dest`. Characters having an ordinal valuelarger than 255 will be replaced by question marks. At most `Len` characters will be copied.

Errors: None.

See also: DefaultAnsi2WideMove (1226)

29.9.42 Delete

Synopsis: Delete part of a string.

Declaration:
```
procedure Delete(var s: shortstring;index: SizeInt;count: SizeInt)
procedure Delete(var S: AnsiString;Index: SizeInt;Size: SizeInt)
procedure Delete(var S: WideString;Index: SizeInt;Size: SizeInt)
```

Visibility: default

Description: `Delete` removes `Count` characters from string `S`, starting at position `Index`. All characters after the delected characters are shifted `Count` positions to the left, and the length of the string is adjusted.

Errors: None.

See also: Copy (1224), Pos (1295), Insert (1255)

Listing: ./refex/ex15.pp

```
Program Example15;

{ Program to demonstrate the Delete function. }

Var
  S : String;

begin
  S:='This is not easy !';
  Delete (S,9,4); { S:='This is easy !' }
end.
```

29.9.43 Dispose

Synopsis: Free dynamically allocated memory

Declaration:
```
procedure Dispose(P: Pointer)
procedure Dispose(P: TypedPointer;Des: TProcedure)
```

Visibility: default

Description: The first form `Dispose` releases the memory allocated with a call to New (1266). The pointer `P` must be typed. The released memory is returned to the heap.

The second form of `Dispose` accepts as a first parameter a pointer to an object type, and as a second parameter the name of a destructor of this object. The destructor will be called, and the memory allocated for the object will be freed.

Errors: An runtime error will occur if the pointer doesn't point to a location in the heap.

See also: New (1266), Getmem (1244), Freemem (1242)

Listing: ./refex/ex16.pp

```
Program Example16;

{ Program to demonstrate the Dispose and New functions. }

Type SS = String[20];

     AnObj = Object
       I : integer;
       Constructor Init;
       Destructor Done;
       end;

Var
  P : ^SS;
  T : ^AnObj;

Constructor Anobj.Init;

begin
 WriteIn ('Initializing an instance of AnObj !');
end;

Destructor AnObj.Done;

begin
  WriteIn ('Destroying an instance of AnObj !');
end;

begin
  New (P);
  P^:='Hello, World !';
  Dispose (P);
  { P is undefined from here on !}
  New(T,Init);
  T^.i:=0;
  Dispose (T,Done);
end.
```

29.9.44 DoneCriticalsection

Synopsis: Clean up a critical section.

Declaration: `procedure DoneCriticalsection(var cs: TRTLCriticalSection)`

Visibility: default

Description: `DoneCriticalsection` cleans up the critical section `CS`. After a call to `DoneCriticalsection`, the critical section can no longer be used with EnterCriticalsection (1230) or LeaveCriticalsection (1260), unless it is again initialized with InitCriticalSection (1254)

See also: InitCriticalsection (1254), EnterCriticalsection (1230), LeaveCriticalsection (1260)

29.9.45 Dseg

Synopsis: Return data segment

Declaration: `function Dseg : Word`

Visibility: default

Description: `DSeg` returns the data segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32 bit compiler.

Errors: None.

See also: CSeg (1225), Seg (1309), Ofs (1268), Ptr (1296)

Listing: ./refex/ex17.pp

```
Program Example17;

{ Program to demonstrate the DSeg function. }

Var
  W : Word;

begin
  W:=DSeg; {W:=0, This function is provided for compatibility,
                  FPC is a 32 bit comiler.}
end.
```

29.9.46 DumpExceptionBackTrace

Synopsis: Create backtrace

Declaration: `procedure DumpExceptionBackTrace(var f: text)`

Visibility: default

Description: `DumpExceptionBackTrace` writes a backtrace of the current exception to the file `f`. If no exception is currently being raised, nothing is written. As much frames as available are written. If debug info is available, then file names and line numbers will be written as well.

Errors: No check is done to see whether `f` is opened for writing.

See also: dump_stack (1229)

29.9.47 Dump_Stack

Synopsis: Dump stack to the given text file.

Declaration: `procedure Dump_Stack(var f: text;bp: pointer)`

Visibility: default

Description: `Dump_Stack` prints a stack dump to the file `f`, with base frame pointer `bp`

Errors: The file `f` must be opened for writing or an error will occur.

See also: get_caller_addr (1246), get_caller_frame (1246), get_frame (1247)

29.9.48 DynArraySetLength

Synopsis: Set the length of a dynamic array

Declaration:
```
procedure DynArraySetLength(var a: Pointer;typeInfo: Pointer;
                            dimCnt: SizeInt;lengthVec: PSizeInt)
```

Visibility: default

Description: `DynArraySetLength` sets the length of the dynamical array `a` to the first `dimCnt` lengths specified in the array `lengthVec`. The dynamical array type is described in `typeInfo` which points to a record of type TDynArrayTypeInfo (1191)

It should never be necessary to call this function directly, the standard SetLength (1311) function should be used instead.

Errors: If an invalid pointer is specified, an error may occur.

See also: SetLength (1311), tdynarraytypeinfo (1191)

29.9.49 EndThread

Synopsis: End the current thread.

Declaration:
```
procedure EndThread(ExitCode: DWord)
procedure EndThread
```

Visibility: default

Description: `EndThread` ends the current thread. If `ExitCode` is supplied, it is returned as the exit code for the thread to a function waiting for the thread to terminate (WaitForThreadTerminate (1332)). If it is omitted, zero is used.

This function does not return.

See also: WaitForThreadTerminate (1332), BeginThread (1212)

29.9.50 EnterCriticalsection

Synopsis: Enter a critical section

Declaration:
```
procedure EnterCriticalsection(var cs: TRTLCriticalSection)
```

Visibility: default

Description: `EnterCriticalSection` will suspend the current thread if another thread has currently entered the critical section. When the other thread has left the critical section (through LeaveCriticalSection (1260)), the current thread resumes execution. The result is that only 1 thread is executing code which is protected by a `EnterCriticalsection` and `LeaveCriticalSection` pair.

The critical section must have been initialized with InitCriticalSection (1254) prior to a call to `EnterCriticalsection`.

A call to `EnterCriticalsection` must always be matched by a call to LeaveCriticalSection (1260). To avoid problems, it is best to include the code to be execute in a `try...finally` block, as follows:

```
EnterCriticalSection(Section);
  Try
    // Code to be protected goes here.
  Finally
    LeaveCriticalSection(Section);
  end;
```

For performance reasons it is best to limit the code between the entering and leaving of a critical section as short as possible.

See also: InitCriticalsection (1254), DoneCriticalsection (1228), LeaveCriticalsection (1260)

29.9.51 EOF

Synopsis: Check for end of file

Declaration:
```
function EOF(var f: File) : Boolean
function EOF(var t: Text) : Boolean
function EOF : Boolean
```

Visibility: default

Description: `Eof` returns `True` if the file-pointer has reached the end of the file, or if the file is empty. In all other cases `Eof` returns `False`. If no file `F` is specified, standard input is assumed.

Note that calling this function may cause your program to wait: to determine whether you are at EOF, it is necessary to read data. If the file descriptor is not a real file (for instance for standard input or sockets) , then this call may seem to hang the program while it is waiting for data to appear or for the file descriptor to be closed.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Eoln (1232), Assign (1210), Reset (1302), Rewrite (1303)

Listing: ./refex/ex18.pp

```
Program Example18;

{ Program to demonstrate the Eof function. }

Var T1,T2 : text;
    C : Char;

begin
  { Set file to read from. Empty means from standard input.}
  assign (t1,paramstr(1));
  reset (t1);
  { Set file to write to. Empty means to standard output. }
  assign (t2,paramstr(2));
  rewrite (t2);
  While not eof(t1) do
    begin
    read (t1,C);
    write (t2,C);
    end;
  Close (t1);
  Close (t2);
end.
```

29.9.52 EOLn

Synopsis: Check for end of line

Declaration:
```
function EOLn(var t: Text) : Boolean
function EOLn : Boolean
```

Visibility: default

Description: `Eof` returns `True` if the file pointer has reached the end of a line, which is demarcated by a line-feed character (ASCII value 10), or if the end of the file is reached. In all other cases `Eof` returns `False`. If no file `F` is specified, standard input is assumed. It can only be used on files of type `Text`.

Errors: None.

See also: Eof (1231), Assign (1210), Reset (1302), Rewrite (1303)

Listing: ./refex/ex19.pp

```
Program Example19;

{ Program to demonstrate the Eoln function. }

begin
  { This program waits for keyboard input. }
  { It will print True when an empty line is put in,
    and false when you type a non-empty line.
    It will only stop when you press enter.}
  While not Eoln do
    Writeln (eoln);
end.
```

29.9.53 Erase

Synopsis: Delete a file from disk

Declaration:
```
procedure Erase(var f: File)
procedure Erase(var t: Text)
```

Visibility: default

Description: `Erase` removes an unopened file from disk. The file should be assigned with `Assign`, but not opened with `Reset` or `Rewrite`

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Assign (1210)

Listing: ./refex/ex20.pp

```
Program Example20;

{ Program to demonstrate the Erase function. }

Var F : Text;

begin
  { Create a file with a line of text in it}
```

```
  Assign (F,'test.txt');
  Rewrite (F);
  WriteIn (F,'Try and find this when I''m finished !');
  close (f);
  { Now remove the file }
  Erase (f);
end.
```

29.9.54 Error

Synopsis: 32-bit signed integer.

Declaration: `procedure Error(RunTimeError: TRuntimeError)`

Visibility: default

29.9.55 Exclude

Synopsis: Exlude element from a set if it is present.

Declaration: `procedure Exclude(var S: TSetType;E: TSetElement)`

Visibility: default

Description: `Exclude` removes `E` from the set `S` if it is included inthe set. E should be of the same type as the base type of the set `S`.

Thus, the two following statements do the same thing:

```
S:=S-[E];
Exclude(S,E);
```

Errors: If the type of the element `E` is not equal to the base type of the set `S`, the compiler will generate an error.

See also: Include (1251)

Listing: ./refex/ex111.pp

```
program Example111;

{ Program to demonstrate the Include/Exclude functions }

Type
  TEnumA = (aOne,aTwo,aThree);
  TEnumAs = Set of TEnumA;

Var
  SA : TEnumAs;

  Procedure PrintSet(S : TEnumAs);

  var
    B : Boolean;

    procedure DoEl(A : TEnumA; Desc : String);
```

```
    begin
      If A in S then
        begin
        If B then
          Write(',');
        B:=True;
        Write(Desc);
        end;
    end;

  begin
    Write('[');
    B:=False;
    DoEl(aOne,'aOne');
    DoEl(aTwo,'aTwo');
    DoEl(aThree,'aThree');
    Writeln(']')
  end;

begin
  SA:=[];
  Include(SA,aOne);
  PrintSet(SA);
  Include(SA,aThree);
  PrintSet(SA);
  Exclude(SA,aOne);
  PrintSet(SA);
  Exclude(SA,aTwo);
  PrintSet(SA);
  Exclude(SA,aThree);
  PrintSet(SA);
end.
```

29.9.56 Exit

Synopsis: Exit current subroutine.

Declaration:
```
procedure Exit(const X: TAnyType)
procedure Exit
```

Visibility: default

Description: `Exit` exits the current subroutine, and returns control to the calling routine. If invoked in the main program routine, exit stops the program. The optional argument `X` allows to specify a return value, in the case `Exit` is invoked in a function. The function result will then be equal to `X`.

Errors: None.

See also: Halt (1247)

Listing: ./refex/ex21.pp

```
Program Example21;

{ Program to demonstrate the Exit function. }

Procedure DoAnExit (Yes : Boolean);
```

```
{ This procedure demonstrates the normal Exit }

begin
  WriteLn ('Hello from DoAnExit !');
  If Yes then
    begin
    WriteLn ('Bailing out early.');
    exit;
    end;
  WriteLn ('Continuing to the end.');
end;

Function Positive (Which : Integer) : Boolean;

{ This function demonstrates the extra FPC feature of Exit :
  You can specify a return value for the function }

begin
  if Which>0 then
    exit (True)
  else
    exit (False);
end;

begin
  { This call will go to the end }
  DoAnExit (False);
  { This call will bail out early }
  DoAnExit (True);
  if Positive (-1) then
    WriteLn ('The compiler is nuts, -1 is not positive.')
  else
    WriteLn ('The compiler is not so bad, -1 seems to be negative.');
end.
```

29.9.57 exp

Synopsis: Exponentiate

Declaration: `function exp(d: ValReal) : ValReal`

Visibility: default

Description: `Exp` returns the exponent of `X`, i.e. the number `e` to the power `X`.

Errors: None.

See also: Ln (1261), Power (1159)

Listing: ./refex/ex22.pp

```
Program Example22;

{ Program to demonstrate the Exp function. }

begin
  WriteLn (Exp(1):8:2); { Should print 2.72 }
end.
```

29.9.58 FilePos

Synopsis: Get position in file

Declaration: `function FilePos(var f: File) : Int64`

Visibility: default

Description: `Filepos` returns the current record position of the file-pointer in file `F`. It cannot be invoked with a file of type `Text`. A compiler error will be generated if this is attempted.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Filesize (1236)

Listing: ./refex/ex23.pp

```
Program Example23;

{ Program to demonstrate the FilePos function. }

Var F : File of Longint;
    L,FP : longint;

begin
  { Fill a file with data :
    Each position contains the position ! }
  Assign (F,'test.tmp');
  Rewrite (F);
  For L:=0 to 100 do
    begin
    FP:=FilePos(F);
    Write (F,FP);
    end;
  Close (F);
  Reset (F);
  { If all goes well, nothing is displayed here. }
  While not (Eof(F)) do
    begin
    FP:=FilePos (F);
    Read (F,L);
    if L<>FP then
      Writeln ('Something wrong: Got ',L,' on pos ',FP);
    end;
  Close (F);
  Erase (f);
end.
```

29.9.59 FileSize

Synopsis: Size of file

Declaration: `function FileSize(var f: File) : Int64`

Visibility: default

Description: `Filesize` returns the total number of records in file `F`. It cannot be invoked with a file of type `Text`. (under linux and unix, this also means that it cannot be invoked on pipes). If `F` is empty, 0 is returned.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Filepos (1236)

Listing: ./refex/ex24.pp

```
Program Example24;

{ Program to demonstrate the FileSize function. }

Var F : File Of byte;
    L : File Of Longint;

begin
  Assign (F,paramstr(1));
  Reset (F);
  Writeln ('File size in bytes : ',FileSize(F));
  Close (F);
  Assign (L,paramstr (1));
  Reset (L);
  Writeln ('File size in Longints : ',FileSize(L));
  Close (f);
end.
```

29.9.60 FillByte

Synopsis: Fill memory region with 8-bit pattern

Declaration: `procedure FillByte(var x;count: SizeInt;value: Byte)`

Visibility: default

Description: `FillByte` fills the memory starting at `X` with `Count` bytes with value equal to `Value`. This is useful for quickly zeroing out a memory location. When the size of the memory location to be filled out is a multiple of 2 bytes, it is better to use Fillword (1239), and if it is a multiple of 4 bytes it is better to use FillDWord (1239), these routines are optimized for their respective sizes.

Errors: No checking on the size of `X` is done.

See also: Fillchar (1238), FillDWord (1239), Fillword (1239), Move (1265)

Listing: ./refex/ex102.pp

```
Program Example102;

{ Program to demonstrate the CompareWord function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of Word;
  I : longint;

  Procedure CheckPos(Len : Longint);
```

```
  Begin
    Write('First ',Len,' words are ');
    if CompareWord(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;


begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=I;
    If I<=HalfArraySize Then
      Buf2[I]:=I
    else
      Buf2[i]:=HalfArraySize-I;
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
end.
```

29.9.61 FillChar

Synopsis: Fill memory region with certain character

Declaration:
```
procedure FillChar(var x;count: SizeInt;Value: Byte)
procedure FillChar(var x;count: SizeInt;Value: Boolean)
procedure FillChar(var x;count: SizeInt;Value: Char)
```

Visibility: default

Description: `Fillchar` fills the memory starting at `X` with `Count` bytes or characters with value equal to `Value`.

Errors: No checking on the size of `X` is done.

See also: Fillword (1239), Move (1265), FillByte (1237), FillDWord (1239)

Listing: ./refex/ex25.pp

```
Program Example25;

{ Program to demonstrate the FillChar function. }

Var S : String[10];
    I : Byte;
begin
  For i:=10 downto 0 do
    begin
    { Fill S with i spaces }
    FillChar (S,SizeOf(S),' ');
    { Set Length }
    SetLength(S,I);
    Writeln (s,'*');
    end;
end.
```

29.9.62 FillDWord

Synopsis: Fill memory region with 32-bit pattern

Declaration: `procedure FillDWord(var x;count: SizeInt;value: DWord)`

Visibility: default

Description: `Fillword` fills the memory starting at `X` with `Count` DWords with value equal to `Value`. A DWord is 4 bytes in size.

Errors: No checking on the size of `X` is done.

See also: FillByte (1237), Fillchar (1238), Fillword (1239), Move (1265)

Listing: ./refex/ex103.pp

```
Program Example103;

{ Program to demonstrate the FillByte function. }

Var S : String[10];
    I : Byte;

begin
  For i:=10 downto 0 do
    begin
    { Fill S with i bytes }
    FillChar (S,SizeOf(S),32);
    { Set Length }
    SetLength(S,I);
    Writeln (s,'*');
    end;
end.
```

29.9.63 FillWord

Synopsis: Fill memory region with 16-bit pattern

Declaration: `procedure FillWord(var x;count: SizeInt;Value: Word)`

Visibility: default

Description: `Fillword` fills the memory starting at `X` with `Count` words with value equal to `Value`. A word is 2 bytes in size.

Errors: No checking on the size of `X` is done.

See also: Fillchar (1238), Move (1265)

Listing: ./refex/ex76.pp

```
Program Example76;

{ Program to demonstrate the FillWord function. }

Var W : Array[1..100] of Word;

begin
```

```
  { Quick initialization of array W }
  FillWord(W,100,0);
end.
```

29.9.64 FindResource

Synopsis: Locate a resource and return a handle to it.

Declaration:
```
function FindResource(ModuleHandle: HMODULE;ResourceName: PChar;
                      ResourceType: PChar) : TResourceHandle
function FindResource(ModuleHandle: HMODULE;ResourceName: AnsiString;
                      ResourceType: AnsiString) : TResourceHandle
```

Visibility: default

Description: `FindResource` searches for a resource with name `ResourceName` and of type `ResourceType` in the executable or library identified by `ModuleHandle`. It returns a `TResourceHandle` which can be used to load the resource with LoadResource (1262).

Errors: None. In case the resource was not found, 0 is returned.

See also: FreeResource (1243), LoadResource (1262), SizeofResource (1316), LockResource (1263), UnlockResource (1329), FreeResource (1243)

29.9.65 float_raise

Synopsis: Raise floating point exception

Declaration:
```
procedure float_raise(i: ShortInt)
```

Visibility: default

Description: `float_raise` raises the floating point exceptions specified by softfloat_exception_flags (1204).

See also: softfloat_exception_flags (1204), softfloat_exception_mask (1204)

29.9.66 Flush

Synopsis: Write file buffers to disk

Declaration:
```
procedure Flush(var t: Text)
```

Visibility: default

Description: `Flush` empties the internal buffer of an opened file `F` and writes the contents to disk. The file is \textit{not} closed as a result of this call.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Close (1217)

Listing: ./refex/ex26.pp

```
Program Example26;

{ Program to demonstrate the Flush function. }

Var F : Text;

begin
  { Assign F to standard output }
  Assign (F,'');
  Rewrite (F);
  Writeln (F,'This line is written first, but appears later !');
  { At this point the text is in the internal pascal buffer,
    and not yet written to standard output }
  Writeln ('This line appears first, but is written later !');
  { A writeln to 'output' always causes a flush - so this text is
    written to screen }
  Flush (f);
  { At this point, the text written to F is written to screen. }
  Write (F,'Finishing ');
  Close (f); { Closing a file always causes a flush first }
  Writeln ('off.');
end.
```

29.9.67 FlushThread

Synopsis: Flush all standard files

Declaration: `procedure FlushThread`

Visibility: default

Description: `FlushThread` flushes any buffers from standard file descriptors such as standard input/output/error. It should normally not be called by user code, but is executed when a thread exits.

See also: EndThread (1230)

29.9.68 frac

Synopsis: Return fractional part of floating point value.

Declaration: `function frac(d: ValReal) : ValReal`

Visibility: default

Description: `Frac` returns the non-integer part of `X`.

Errors: None.

See also: Round (1305), Int (1256)

Listing: ./refex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Frac function. }

Var R : Real;
```

```
begin
  WriteLn (Frac (123.456):0:3);  { Prints  0.456 }
  WriteLn (Frac (-123.456):0:3); { Prints -0.456 }
end.
```

29.9.69 Freemem

Synopsis: Release allocated memory

Declaration:
```
procedure Freemem(p: pointer;Size: PtrInt)
function Freemem(p: pointer) : PtrInt
```

Visibility: default

Description: `Freemem` releases the memory occupied by the pointer `P`, of size `Count` (in bytes), and returns it to the heap. `P` should point to the memory allocated to a dynamic variable.

Errors: An error will occur when `P` doesn't point to the heap.

See also: Getmem (1244), New (1266), Dispose (1227)

Listing: ./refex/ex28.pp

```
Program Example28;

{ Program to demonstrate the FreeMem and GetMem functions. }

Var P : Pointer;
    MM : Longint;

begin
  { Get memory for P }
  GetMem (P,80);
  FillChar (P^,80,' ');
  FreeMem (P,80);
end.
```

29.9.70 Freememory

Synopsis: Alias for FreeMem (1242)

Declaration:
```
procedure Freememory(p: pointer;Size: PtrInt)
function Freememory(p: pointer) : PtrInt
```

Visibility: default

Description: `FreeMemory` is an alias for FreeMem (1242).

See also: FreeMem (1242)

29.9.71 FreeResource

Synopsis: Free a loaded resource

Declaration: `function FreeResource(ResData: HGLOBAL) : LongBool`

Visibility: default

Description: `FreeResource` unloads the resource identified by `ResData` from memory. The resource must have been loaded by LoadResource (1262). It returns `True` if the operation was succesful, `False` otherwise.

Errors: On error, `False` is returned.

See also: FindResource (1240), LoadResource (1262), SizeofResource (1316), LockResource (1263), UnlockResource (1329), FreeResource (1243)

29.9.72 GetCurrentThreadId

Synopsis: Return the id of the currently running thread.

Declaration: `function GetCurrentThreadId : TThreadID`

Visibility: default

Description: `GetCurrentThreadId` returns the ID of the currently running thread. It can be used in calls such as KillThread (1260) or ThreadSetPriority (1326)

Errors: None.

See also: KillThread (1260), ThreadSetPriority (1326)

29.9.73 getdir

Synopsis: Return the current directory

Declaration:
```
procedure getdir(drivenr: Byte;var dir: shortstring)
procedure getdir(drivenr: Byte;var dir: ansistring)
```

Visibility: default

Description: `Getdir` returns in `dir` the current directory on the drive `drivenr`, where {drivenr} is 1 for the first floppy drive, 3 for the first hard disk etc. A value of 0 returns the directory on the current disk. On linux and unix systems, `drivenr` is ignored, as there is only one directory tree.

Errors: An error is returned under dos, if the drive requested isn't ready.

See also: Chdir (1216)

Listing: ./refex/ex29.pp

```
Program Example29;

{ Program to demonstrate the GetDir function. }

Var S : String;

begin
  GetDir (0,S);
  Writeln ('Current directory is : ',S);
end.
```

29.9.74 GetFPCHeapStatus

Synopsis: Return FPC heap manager status information

Declaration:
```
function GetFPCHeapStatus : TFPCHeapStatus
```

Visibility: default

Description: Return FPC heap manager status information

Errors:

29.9.75 GetHeapStatus

Synopsis: Return the memory manager heap status.

Declaration:
```
function GetHeapStatus : THeapStatus
```

Visibility: default

29.9.76 GetMem

Synopsis: Allocate new memory on the heap

Declaration:
```
procedure Getmem(var p: pointer;Size: PtrInt)
function GetMem(size: PtrInt) : pointer
```

Visibility: default

Description: `Getmem` reserves `Size` bytes memory on the heap, and returns a pointer to this memory in `p`. If no more memory is available, nil is returned.

For an example, see Freemem (1242).

Errors: None.

See also: Freemem (1242), Dispose (1227), New (1266)

29.9.77 GetMemory

Synopsis: Alias for GetMem (1244)

Declaration:
```
procedure Getmemory(var p: pointer;Size: PtrInt)
function GetMemory(size: PtrInt) : pointer
```

Visibility: default

Description: `Getmemory` is an alias for GetMem (1244).

See also: GetMem (1244)

29.9.78 GetMemoryManager

Synopsis: Return current memory manager

Declaration: `procedure GetMemoryManager(var MemMgr: TMemoryManager)`

Visibility: default

Description: `GetMemoryManager` stores the current Memory Manager record in `MemMgr`.

For an example, see \progref.

Errors: None.

See also: SetMemoryManager (1311), IsMemoryManagerSet (1259)

29.9.79 GetProcessID

Synopsis: Get the current process ID

Declaration: `function GetProcessID : SizeUInt`

Visibility: default

Description: `GetProcessID` returns the current process ID. The meaning of the return value of this call is system dependent.

Errors: None.

See also: GetThreadID (1245)

29.9.80 GetThreadID

Synopsis: Get the current Thread ID.

Declaration: `function GetThreadID : TThreadID`

Visibility: default

Description: `GetThreadID` returns the current process ID. The meaning of the return value of this call is system dependent.

See also: GetProcessID (1245)

29.9.81 GetThreadManager

Synopsis: Return the current thread manager

Declaration: `function GetThreadManager(var TM: TThreadManager) : Boolean`

Visibility: default

Description: `GetThreadManager` returns the currently used thread manager in `TM`.

For more information about thread programming, see the programmer's guide.

See also: SetThreadManager (1314), TThreadManager (1198)

29.9.82 GetVariantManager

Synopsis: Return the current variant manager.

Declaration: `procedure GetVariantManager(var VarMgr: tvariantmanager)`

Visibility: default

Description: `GetVariantManager` returns the current variant manager in `varmgr`.

See also: IsVariantManagerSet (1159), SetVariantManager (1314)

29.9.83 GetWideStringManager

Synopsis: Return a copy of the currently active widestring manager.

Declaration: `procedure GetWideStringManager(var Manager: TWideStringManager)`

Visibility: default

Description: `GetWideStringManager` returns a copy of the currently active heap manager in `Old`

WideStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a WideString manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit cwstrings).

It is possible to implement a custom widestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: SetWideStringManager (1314), TWideStringManager (1202)

29.9.84 get_caller_addr

Synopsis: Return the address of the caller.

Declaration: `function get_caller_addr(framebp: pointer) : pointer`

Visibility: default

Description: `get_caller_frame` returns a pointer to address (the return address) of the caller of the routine which has as frame `framebp`.

See also: get_frame (1247), get_caller_frame (1246), Dump_Stack (1229)

29.9.85 get_caller_frame

Synopsis: Return the frame pointer of the caller

Declaration: `function get_caller_frame(framebp: pointer) : pointer`

Visibility: default

Description: `get_caller_frame` returns a pointer to the frame of the caller of the routine which has as frame `framebp`.

See also: get_caller_addr (1246), get_frame (1247), Dump_Stack (1229)

29.9.86 get_frame

Synopsis: Return the current frame

Declaration:
```
function get_frame : pointer
```

Visibility: default

Description: `get_frame` returns a pointer to the current stack frame.

See also: get_caller_addr (1246), get_caller_frame (1246)

29.9.87 halt

Synopsis: Stop program execution.

Declaration:
```
procedure halt(errnum: Byte)
procedure halt
```

Visibility: default

Description: `Halt` stops program execution and returns control to the calling program. The optional argument `Errnum` specifies an exit value. If omitted, zero is returned.

Errors: None.

See also: Exit (1234)

Listing: ./refex/ex30.pp

```
Program Example30;

{ Program to demonstrate the Halt function. }

begin
 WriteLn ('Before Halt.');
 Halt (1); { Stop with exit code 1 }
 WriteLn ('After Halt doesn''t get executed.');
end.
```

29.9.88 hexStr

Synopsis: Convert integer value to string with hexadecimal representation.

Declaration:
```
function hexStr(Val: LongInt;cnt: Byte) : shortstring
function hexStr(Val: Int64;cnt: Byte) : shortstring
function hexStr(Val: qword;cnt: Byte) : shortstring
function hexStr(Val: Pointer) : shortstring
```

Visibility: default

Description: `HexStr` returns a string with the hexadecimal representation of `Value`. The string has exactly `cnt` charaters. (i.e. only the `cnt` rightmost nibbles are taken into account) To have a complete representation of a Longint-type value, 8 nibbles are needed, i.e. `cnt=8`.

Errors: None.

See also: Str (1318), Val (1331), BinStr (1213)

Listing: ./refex/ex81.pp

```
Program example81;

{ Program to demonstrate the HexStr function }

Const Value = 45678;

Var I : longint;

begin
  For I:=1 to 10 do
    WriteIn (HexStr(Value,I));
end.
```

29.9.89 hi

Synopsis: Return high byte/word of value.

Declaration:
```
function hi(b: Byte) : Byte
function hi(i: Integer) : Byte
function hi(w: Word) : Byte
function hi(l: LongInt) : Word
function hi(l: DWord) : Word
function hi(i: Int64) : DWord
function hi(q: QWord) : DWord
```

Visibility: default

Description: `Hi` returns the high byte or word from `X`, depending on the size of X. If the size of X is 4, then the high word is returned. If the size is 2 then the high byte is returned. `Hi` cannot be invoked on types of size 1, such as byte or char.

Errors: None

See also: Lo (1262)

Listing: ./refex/ex31.pp

```
Program Example31;

{ Program to demonstrate the Hi function. }

var
  L : Longint;
  W : Word;

begin
  L:=1 Shl 16;      { = $10000 }
  W:=1 Shl 8;       { = $100 }
  WriteIn (Hi(L)); { Prints 1 }
  WriteIn (Hi(W)); { Prints 1 }
end.
```

29.9.90 High

Synopsis: Return highest index of open array or enumerated

Declaration: `function High(Arg: TypeOrVariable) : TOrdinal`

Visibility: default

Description: The return value of `High` depends on it's argument:

1. If the argument is an ordinal type, `High` returns the highest value in the range of the given ordinal type.
2. If the argument is an array type or an array type variable then `High` returns the highest possible value of it's index.
3. If the argument is an open array identifier in a function or procedure, then `High` returns the highest index of the array, as if the array has a zero-based index.
4. If the argument is a set type then it returns the highest value of the underlying ordinal type.

The return type is always the same type as the type of the argument (This can lead to some nasty surprises !).

Errors: None.

See also: Low (1263), Ord (1292), Pred (1296), Succ (1321)

Listing: ./refex/ex80.pp

```
Program example80;

{ Example to demonstrate the High and Low functions. }

Type TEnum = ( North, East, South, West );
     TRange = 14..55;
     TArray = Array [2..10] of Longint;

Function Average (Row : Array of Longint) : Real;

Var I : longint;
    Temp : Real;


begin
  Temp := Row[0];
  For I := 1 to High(Row) do
    Temp := Temp + Row[i];
  Average := Temp / (High(Row)+1);
end;

Var A : TEnum;
    B : TRange;
    C : TArray;
    I : longint;

begin
  Writeln ('TEnum  goes from : ',Ord(Low(TEnum)),' to ', Ord(high(TEnum)),'.');
  Writeln ('A      goes from : ',Ord(Low(A)),' to ', Ord(high(A)),'.');
  Writeln ('TRange goes from : ',Ord(Low(TRange)),' to ', Ord(high(TRange)),'.');
  Writeln ('B      goes from : ',Ord(Low(B)),' to ', Ord(high(B)),'.');
```

```
  WriteLn ('TArray index goes from : ',Ord(Low(TArray)),' to ', Ord(high(TArray)),'.');
  WriteLn ('C index      goes from : ',Low(C),' to ', high(C),'.');
  For I:=Low(C) to High(C) do
    C[i]:=I;
  WriteLn ('Average :',Average(c));
  Write ('Type of return value is always same as type of argument:');
  WriteLn(high(high(word)));
end.
```

29.9.91 HINSTANCE

Synopsis: Windows compatibility type for use in resources

Declaration: `function HINSTANCE : HMODULE`

Visibility: default

Description: This is an opaque type.

29.9.92 Inc

Synopsis: Increase value of integer variable

Declaration:
```
procedure Inc(var X: TOrdinal)
procedure Inc(var X: TOrdinal;Increment: TOrdinal)
```

Visibility: default

Description: `Inc` increases the value of `X` with `Increment`. If `Increment` isn't specified, then 1 is taken as a default.

Errors: If range checking is on, then A range check can occur, or an overflow error, when an attempt is made to increase `X` over its maximum value.

See also: Dec (1225)

Listing: ./refex/ex32.pp

```
Program Example32;

{ Program to demonstrate the Inc function. }

Const
  C : Cardinal  = 1;
  L : Longint   = 1;
  I : Integer   = 1;
  W : Word      = 1;
  B : Byte      = 1;
  SI : ShortInt = 1;
  CH : Char     = 'A';

begin
  Inc (C);      { C:=2    }
  Inc (L,5);    { L:=6    }
  Inc (I,-3);   { I:=-2   }
  Inc (W,3);    { W:=4    }
  Inc (B,100); { B:=101  }
```

```
  Inc (SI,-3); { Si:=-2 }
  Inc (CH,1);  { ch:='B' }
end.
```

29.9.93 Include

Synopsis: Include element in set if it was not yet present.

Declaration: `procedure Include(var S: TSetType;E: TSetElement)`

Visibility: default

Description: `Include` includes `E` in the set `S` if it is not yet part of the set. E should be of the same type as the base type of the set `S`.

Thus, the two following statements do the same thing:

```
S:=S+[E];
Include(S,E);
```

For an example, see Exclude (1233)

Errors: If the type of the element `E` is not equal to the base type of the set `S`, the compiler will generate an error.

See also: Exclude (1233)

29.9.94 IndexByte

Synopsis: Search for a byte in a memory range.

Declaration: `function IndexByte(const buf;len: SizeInt;b: Byte) : SizeInt`

Visibility: default

Description: `IndexByte` searches the memory at `buf` for maximally `len` positions for the byte `b` and returns it's position if it found one. If `b` is not found then -1 is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexChar (1252), IndexDWord (1253), IndexWord (1254), CompareByte (1217)

Listing: ./refex/ex105.pp

```
Program Example105;

{ Program to demonstrate the IndexByte function. }

Const
  ArraySize = 256;
  MaxValue = 256;

Var
  Buffer : Array[1..ArraySize] of Byte;
  I,J : longint;
  K : Byte;
```

```
begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
    K:=Random(MaxValue);
    J:=IndexByte(Buffer,ArraySize,K);
    if J=-1 then
      Writeln('Value ',K,' was not found in buffer.')
    else
      Writeln('Found ',K,' at position ',J,' in buffer');
    end;
end.
```

29.9.95 IndexChar

Synopsis: Search for a character in a memory range.

Declaration: `function IndexChar(const buf;len: SizeInt;b: Char) : SizeInt`

Visibility: default

Description: `IndexChar` searches the memory at `buf` for maximally `len` positions for the character `b` and returns it's position if it found one. If `b` is not found then -1 is returned. The position is zero-based. The `IndexChar0` variant stops looking if a null character is found, and returns -1 in that case.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexByte (1251), IndexDWord (1253), IndexWord (1254), CompareChar (1218)

Listing: ./refex/ex108.pp

```
Program Example108;

{ Program to demonstrate the IndexChar function. }

Const
  ArraySize = 1000;
  MaxValue = 26;

Var
  Buffer : Array[1..ArraySize] of Char;
  I,J : longint;
  K : Char;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=chr(Ord('A')+Random(MaxValue));
  For I:=1 to 10 do
    begin
    K:=chr(Ord('A')+Random(MaxValue));
    J:=IndexChar(Buffer,ArraySize,K);
    if J=-1 then
      Writeln('Value ',K,' was not found in buffer.')
    else
      Writeln('Found ',K,' at position ',J,' in buffer');
```

```
    end;
end.
```

29.9.96 IndexChar0

Synopsis: Return index of a character in null-terminated array of char.

Declaration: `function IndexChar0(const buf;len: SizeInt;b: Char) : SizeInt`

Visibility: default

Description: `IndexChar0` returns the index of the character `b` in the null-terminated array `Buf`. At most `len` characters will be searched, or the null character if it is encountered first. If the character is not found, 0 is returned.

Errors: On error, 0 is returned.

See also: IndexByte (1251), IndexChar (1252), IndexWord (1254), IndexDWord (1253), CompareChar0 (1220)

29.9.97 IndexDWord

Synopsis: Search for a DWord value in a memory range.

Declaration: `function IndexDWord(const buf;len: SizeInt;b: DWord) : SizeInt`

Visibility: default

Description: `IndexChar` searches the memory at `buf` for maximally `len` positions for the DWord `DW` and returns it's position if it found one. If `DW` is not found then -1 is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexByte (1251), IndexChar (1252), IndexWord (1254), CompareDWord (1220)

Listing: ./refex/ex106.pp

```
Program Example106;

{ Program to demonstrate the IndexDWord function. }

Const
  ArraySize = 1000;
  MaxValue = 1000;

Var
  Buffer : Array[1..ArraySize] of DWord;
  I,J : longint;
  K : DWord;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
    K:=Random(MaxValue);
    J:=IndexDWord(Buffer,ArraySize,K);
    if J=-1 then
```

```
      WriteIn('Value ',K,' was not found in buffer.')
    else
      WriteIn('Found ',K,' at position ',J,' in buffer');
    end;
end.
```

29.9.98 Indexword

Synopsis: Search for a WORD value in a memory range.

Declaration: `function Indexword(const buf;len: SizeInt;b: Word) : SizeInt`

Visibility: default

Description: `IndexChar` searches the memory at `buf` for maximally `len` positions for the Word `W` and returns it's position if it found one. If `W` is not found then -1 is returned.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexByte (1251), IndexDWord (1253), IndexChar (1252), CompareWord (1221)

Listing: ./refex/ex107.pp

```
Program Example107;

{ Program to demonstrate the IndexWord function. }

Const
  ArraySize = 1000;
  MaxValue = 1000;

Var
  Buffer : Array[1..ArraySize] of Word;
  I,J : longint;
  K : Word;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
    K:=Random(MaxValue);
    J:=IndexWord(Buffer,ArraySize,K);
    if J=-1 then
      WriteIn('Value ',K,' was not found in buffer.')
    else
      WriteIn('Found ',K,' at position ',J,' in buffer');
    end;
end.
```

29.9.99 InitCriticalSection

Synopsis: Initialize a critical section

Declaration: `procedure InitCriticalSection(var cs: TRTLCriticalSection)`

Visibility: default

Description: `InitCriticalSection` initializes a critical section `CS` for use. Before using a critical section with EnterCriticalsection (1230) or LeaveCriticalsection (1260) the critical section should be initialized with `InitCriticalSection`.

When a critical section is no longer used, it should be disposed of with DoneCriticalsection (1228)

See also: DoneCriticalsection (1228), EnterCriticalsection (1230), LeaveCriticalsection (1260)

29.9.100 InitThread

Synopsis: Initialize a thread

Declaration:
```
procedure InitThread(stklen: SizeUInt)
```

Visibility: default

Description: Do not use, this is used internally by the thread manager.

29.9.101 InitThreadVars

Synopsis: Initialize threadvars

Declaration:
```
procedure InitThreadVars(RelocProc: Pointer)
```

Visibility: default

Description: This routine should be called when threading is started. It is called by the compiler and should never be called manually, only from a thread manager.

Errors: None.

See also: TThreadManager (1198), TThreadManager.InitThreadVar (1198)

29.9.102 Insert

Synopsis: Insert one string in another.

Declaration:
```
procedure Insert(const source: shortstring;var s: shortstring;
                 index: SizeInt)
procedure Insert(source: Char;var s: shortstring;index: SizeInt)
procedure Insert(const Source: AnsiString;var S: AnsiString;
                 Index: SizeInt)
procedure Insert(const Source: WideString;var S: WideString;
                 Index: SizeInt)
```

Visibility: default

Description: `Insert` inserts string `Source` in string `S`, at position `Index`, shifting all characters after `Index` to the right. The resulting string is truncated at 255 characters, if needed. (i.e. for shortstrings)

Errors: None.

See also: Delete (1227), Copy (1224), Pos (1295)

Listing: ./refex/ex33.pp

```
Program Example33;

{ Program to demonstrate the Insert function. }

Var S : String;

begin
  S:='Free Pascal is difficult to use !';
  Insert ('NOT ',S,pos('difficult',S));
  writeln (s);
end.
```

29.9.103 int

Synopsis: Calculate integer part of floating point value.

Declaration:
```
function int(d: ValReal) : ValReal
```

Visibility: default

Description: `Int` returns the integer part of any Real `X`, as a Real.

Errors: None.

See also: Frac (1241), Round (1305)

Listing: ./refex/ex34.pp

```
Program Example34;

{ Program to demonstrate the Int function. }

begin
  Writeln (Int(123.456):0:1); { Prints  123.0 }
  Writeln (Int(-123.456):0:1); { Prints -123.0 }
end.
```

29.9.104 InterlockedCompareExchange

Synopsis: Conditional exchange

Declaration:
```
function InterlockedCompareExchange(var Target: LongInt;
                                    NewValue: LongInt;Comperand: LongInt)
                                     : LongInt
function InterlockedCompareExchange(var Target: Pointer;
                                    NewValue: Pointer;Comperand: Pointer)
                                     : Pointer
```

Visibility: default

Description: `InterlockedCompareExchange` does an compare-and-exchange operation on the specified values in a thread-safe way. The function compares `Target` and `Comparand` and exchanges `Target` with `NewValue` if `Target` and `Comparand` are equal. It returns the old value of `Target`. This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: InterLockedDecrement (1257), InterLockedIncrement (1258), InterLockedExchange (1257), InterLockedExchangeAdd (1257)

29.9.105 InterLockedDecrement

Synopsis: Thread-safe decrement

Declaration:
```
function InterLockedDecrement(var Target: LongInt) : LongInt
function InterLockedDecrement(var Target: Pointer) : Pointer
```

Visibility: default

Description: `InterLockedDecrement` decrements `Target` with 1 and returns the result. This is done in a thread-safe way. (i.e. only one processor is accessing the variable at a time).

Errors: None.

See also: InterLockedIncrement (1258), InterLockedExchange (1257), InterLockedExchangeAdd (1257), InterlockedCompareExchange (1256)

29.9.106 InterLockedExchange

Synopsis: Exchange 2 integers in a thread-safe way

Declaration:
```
function InterLockedExchange(var Target: LongInt;Source: LongInt)
                                : LongInt
function InterLockedExchange(var Target: Pointer;Source: Pointer)
                                : Pointer
```

Visibility: default

Description: `InterLockedExchange` stores `Source` in `Target` and returns the old value of `Target`. This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: InterLockedDecrement (1257), InterLockedIncrement (1258), InterLockedExchangeAdd (1257), InterlockedCompareExchange (1256)

29.9.107 InterLockedExchangeAdd

Synopsis: Thread-safe add and exchange of 2 values

Declaration:
```
function InterLockedExchangeAdd(var Target: LongInt;Source: LongInt)
                                   : LongInt
function InterLockedExchangeAdd(var Target: Pointer;Source: Pointer)
                                   : Pointer
```

Visibility: default

Description: `InterlockedDecrement` adds to `Target` the value of `Source` in a thread-safe way, and returns the old value of `Target`.This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: InterLockedDecrement (1257), InterLockedIncrement (1258), InterLockedExchange (1257), InterlockedCompareExchange (1256)

29.9.108 InterLockedIncrement

Synopsis: Thread-safe increment

Declaration:
```
function InterLockedIncrement(var Target: LongInt) : LongInt
function InterLockedIncrement(var Target: Pointer) : Pointer
```

Visibility: default

Description: `InterLockedIncrement` increments `Target` with 1 and returns the result. This is done in a thread-safe way (i.e. only one processor is accessing the variable at a time).

Errors: None.

See also: InterLockedDecrement (1257), InterLockedExchange (1257), InterLockedExchangeAdd (1257), InterlockedCompareExchange (1256)

29.9.109 IOResult

Synopsis: Return result of last file IO operation

Declaration:
```
function IOResult : Word
```

Visibility: default

Description: IOresult contains the result of any input/output call, when the `{\$i-}` compiler directive is active, disabling IO checking. When the flag is read, it is reset to zero. If `IOresult` is zero, the operation completed successfully. If non-zero, an error occurred. The following errors can occur:

dos errors :

2File not found.

3Path not found.

4Too many open files.

5Access denied.

6Invalid file handle.

12Invalid file-access mode.

15Invalid disk number.

16Cannot remove current directory.

17Cannot rename across volumes.

I/O errors :

100 Error when reading from disk.

101 Error when writing to disk.

102 File not assigned.

103 File not open.

104 File not opened for input.

105 File not opened for output.

106 Invalid number.

Fatal errors :

150 Disk is write protected.

151 Unknown device.

152 Drive not ready.

153 Unknown command.

154CRC check failed.

155 Invalid drive specified..

156 Seek error on disk.

157 Invalid media type.

158 Sector not found.

159 Printer out of paper.

160 Error when writing to device.

161 Error when reading from device.

162 Hardware failure.

Errors: None.

Listing: ./refex/ex35.pp

```
Program Example35;

{ Program to demonstrate the IOResult function. }

Var F : text;

begin
  Assign (f,paramstr(1));
  {$i-}
  Reset (f);
  {$i+}
  If IOresult<>0 then
    writeln ('File ',paramstr(1),' doesn''t exist')
  else
    writeln ('File ',paramstr(1),' exists');
end.
```

29.9.110 IsMemoryManagerSet

Synopsis: Is the memory manager set

Declaration: `function IsMemoryManagerSet : Boolean`

Visibility: default

Description: `IsMemoryManagerSet` will return `True` if the memory manager has been set to another value than the system heap manager, it will return `False` otherwise.

Errors: None.

See also: SetMemoryManager (1311), GetMemoryManager (1245)

29.9.111 KillThread

Synopsis: Kill a running thread

Declaration: `function KillThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `KillThread` causes a running thread to be aborted. The thread is identified by it's handle or ID `threadHandle`.

The function returns zero if succesful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: WaitForThreadTerminate (1332), EndThread (1230), SuspendThread (1321)

29.9.112 LeaveCriticalsection

Synopsis: Leave a critical section

Declaration: `procedure LeaveCriticalsection(var cs: TRTLCriticalSection)`

Visibility: default

Description: `LeaveCriticalSection` signals that the current thread is exiting the critical section `CS` it has entered with EnterCriticalSection (1230).

The critical section must have been initialized with InitCriticalSection (1254) prior to a call to `EnterCriticalsection` and `LeaveCriticalSection`.

See also: InitCriticalsection (1254), DoneCriticalsection (1228), EnterCriticalsection (1230)

29.9.113 Length

Synopsis: Calculate length of a string.

Declaration:
```
function Length(S: AStringType) : Integer
function Length(A: DynArrayType) : Integer
```

Visibility: default

Description: `Length` returns the length of the string `S`, which is limited to 255 for shortstrings. If the strings `S` is empty, 0 is returned.

Note: The length of the string `S` is stored in `S[0]` for shortstrings only. The `Length` fuction should always be used on ansistrings and widestrings.

For dynamical arrays, the function returns the number of elements in the array.

Errors: None.

See also: Pos (1295)

Listing: ./refex/ex36.pp

```
Program Example36;

{ Program to demonstrate the Length function. }

Var S : String;
    I : Integer;
```

```
begin
  S:='';
  for i:=1 to 10 do
    begin
    S:=S+'*';
    Writeln (Length(S):2,' : ',s);
    end;
end.
```

29.9.114 LEtoN

Synopsis: Convert Little Endian-ordered integer to Native-ordered integer

Declaration:
```
function LEtoN(const AValue: SmallInt) : SmallInt
function LEtoN(const AValue: Word) : Word
function LEtoN(const AValue: LongInt) : LongInt
function LEtoN(const AValue: DWord) : DWord
function LEtoN(const AValue: Int64) : Int64
function LEtoN(const AValue: QWord) : QWord
```

Visibility: default

Description: `LEToN` will rearrange the bytes in a Little-Endian number to the native order for the current processor. That is, for a little-endian processor, it will do nothing, and for a big-endian processor, it will invert the order of the bytes.

See also: BEtoN (1213), NtoBE (1266), NtoLE (1267)

29.9.115 ln

Synopsis: Calculate logarithm

Declaration:
```
function ln(d: ValReal) : ValReal
```

Visibility: default

Description: `Ln` returns the natural logarithm of the Real parameter `X`. `X` must be positive.

Errors: An run-time error will occur when `X` is negative.

See also: Exp (1235), Power (1159)

Listing: ./refex/ex37.pp

```
Program Example37;

{ Program to demonstrate the Ln function. }

begin
  Writeln (Ln(1));       { Prints 0 }
  Writeln (Ln(Exp(1))); { Prints 1 }
end.
```

29.9.116 lo

Synopsis: Return low byte/word of value.

Declaration:
```
function lo(B: Byte) : Byte
function lo(i: Integer) : Byte
function lo(w: Word) : Byte
function lo(l: LongInt) : Word
function lo(l: DWord) : Word
function lo(i: Int64) : DWord
function lo(q: QWord) : DWord
```

Visibility: default

Description: `Lo` returns the low byte of its argument if this is of type `Integer` or `Word`. It returns the low word of its argument if this is of type `Longint` or `Cardinal`.

Errors: None.

See also: Ord (1292), Chr (1216), Hi (1248)

Listing: ./refex/ex38.pp

```
Program Example38;

{ Program to demonstrate the Lo function. }

Var L : Longint;
    W : Word;

begin
  L:=(1 Shl 16) + (1 Shl 4);  { $10010 }
  Writeln (Lo(L));            { Prints 16 }
  W:=(1 Shl 8) + (1 Shl 4);   { $110   }
  Writeln (Lo(W));            { Prints 16 }
end.
```

29.9.117 LoadResource

Synopsis: Load a resource for use

Declaration:
```
function LoadResource(ModuleHandle: HMODULE;ResHandle: TResourceHandle)
                      : HGLOBAL
```

Visibility: default

Description: `LoadResource` loads a resource identified by `ResHandle` from a module identified by `ModuleHandle`into memory. It returns a handle to the resource.

Loaded resources must be unloaded again using the FreeResource (1243) function.

Errors: On error, 0 is returned.

See also: FindResource (1240), FreeResource (1243), SizeofResource (1316), LockResource (1263), UnlockResource (1329), FreeResource (1243)

29.9.118 LockResource

Synopsis: Lock a resource

Declaration: `function LockResource(ResData: HGLOBAL) : Pointer`

Visibility: default

Description: `LockResource` locks a resource previously loaded by `LoadResource` into memory. This means that any attempt to modify the resource will fail while it is locked. The function returns a pointer to the resource location in memory.

The resource can be freed again using the UnlockResource (1329) function.

Errors: if the function fails, `Nil` is returned.

See also: FindResource (1240), FreeResource (1243), SizeofResource (1316), LoadResource (1262), UnlockResource (1329), FreeResource (1243)

29.9.119 longjmp

Synopsis: Jump to address.

Declaration: `procedure longjmp(var S: jmp_buf;value: LongInt)`

Visibility: default

Description: `LongJmp` jumps to the adress in the `envjmp_buf`, and restores the registers that were stored in it at the corresponding SetJmp (1310) call. In effect, program flow will continue at the `SetJmp` call, which will return `value` instead of 0. If a `value` equal to zero is passed, it will be converted to 1 before passing it on. The call will not return, so it must be used with extreme care. This can be used for error recovery, for instance when a segmentation fault occurred.

For an example, see SetJmp (1310)

Errors: None.

See also: SetJmp (1310)

29.9.120 Low

Synopsis: Return lowest index of open array or enumerated

Declaration: `function Low(Arg: TypeOrVariable) : TOrdinal`

Visibility: default

Description: The return value of `Low` depends on it's argument:

1. If the argument is an ordinal type, `Low` returns the lowest value in the range of the given ordinal type.
2. If the argument is an array type or an array type variable then `Low` returns the lowest possible value of it's index.
3. If the argument is an open array identifier in a function or procedure, then `Low` returns the lowest element of the array, which is always zero.
4. If the argument is a set type then it returns the lowest value of the underlying ordinal type.

The return type is always the same type as the type of the argument.

for an example, see High (1249).

Errors: None.

See also: High (1249), Ord (1292), Pred (1296), Succ (1321)

29.9.121 lowerCase

Synopsis: Return lowercase version of a string.

Declaration:
```
function lowerCase(const s: shortstring) : shortstring;  Overload
function lowerCase(c: Char) : Char;  Overload
function lowercase(const s: ansistring) : ansistring
```

Visibility: default

Description: `Lowercase` returns the lowercase version of its argument `C`. If its argument is a string, then the complete string is converted to lowercase. The type of the returned value is the same as the type of the argument.

Errors: None.

See also: Upcase (1329)

Listing: ./refex/ex73.pp

```
program Example73;

{ Program to demonstrate the Lowercase function. }

var c:char;

begin
  for c:='A' to 'Z' do
    write(lowercase(c));
  Writeln;
  Writeln(Lowercase('ABCDEFGHIJKLMNOPQRSTUVWXYZ'));
end.
```

29.9.122 MemSize

Synopsis: Return the size of a memory block.

Declaration:
```
function MemSize(p: pointer) : PtrInt
```

Visibility: default

Description: `MemSize` returns the size of a memory block on the heap.

Errors: Passing an invalid pointer may lead to run-time errors (access violations).

See also: GetMem (1244), FreeMem (1242)

29.9.123 mkdir

Synopsis: Create a new directory.

Declaration:
```
procedure mkdir(const s: String)
```

Visibility: default

Description: `Mkdir` creates a new directory `S`.

For an example, see Rmdir (1304).

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Chdir (1216), Rmdir (1304)

29.9.124 Move

Synopsis: Move data from one location in memory to another

Declaration: `procedure Move(const source;var dest;count: SizeInt)`

Visibility: default

Description: `Move` moves `Count` bytes from `Source` to `Dest`.

Errors: If either `Dest` or `Source` is outside the accessible memory for the process, then a run-time error will be generated.

See also: Fillword (1239), Fillchar (1238)

Listing: ./refex/ex42.pp

```
Program Example42;

{ Program to demonstrate the Move function. }

Var S1,S2 : String [30];

begin
  S1:='Hello World !';
  S2:='Bye, bye     !';
  Move (S1,S2,Sizeof(S1));
  Writeln (S2);
end.
```

29.9.125 MoveChar0

Synopsis: Move data till first zero character

Declaration: `procedure MoveChar0(const buf1;var buf2;len: SizeInt)`

Visibility: default

Description: `MoveChar0` moves `Count` bytes from `Src` to `Dest`, and stops moving if a zero character is found.

Errors: No checking is done to see if `Count` stays within the memory allocated to the process.

See also: Move (1265)

Listing: ./refex/ex109.pp

```
Program Example109;

{ Program to demonstrate the MoveChar0 function. }

Var
  Buf1,Buf2 : Array[1..80] of char;
  I : longint;

begin
  Randomize;
  For I:=low(buf1) to high(buf1) do
    Buf1[i]:=chr(Random(16)+Ord('A'));
  Writeln('Original buffer');
  writeln(Buf1);
  Buf1[Random(80)+1]:=#0;
  MoveChar0(Buf1,Buf2,80);
  Writeln('Randomly zero-terminated Buffer');
  Writeln(Buf2);
end.
```

29.9.126 New

Synopsis: Dynamically allocate memory for variable

Declaration:
```
procedure New(var P: Pointer)
procedure New(var P: Pointer;Cons: TProcedure)
```

Visibility: default

Description: `New` allocates a new instance of the type pointed to by `P`, and puts the address in `P`. If P is an object, then it is possible to specify the name of the constructor with which the instance will be created.

For an example, see Dispose (1227).

Errors: If not enough memory is available, `Nil` will be returned.

See also: Dispose (1227), Freemem (1242), Getmem (1244)

29.9.127 NtoBE

Synopsis: Convert Native-ordered integer to a Big Endian-ordered integer

Declaration:
```
function NtoBE(const AValue: SmallInt) : SmallInt
function NtoBE(const AValue: Word) : Word
function NtoBE(const AValue: LongInt) : LongInt
function NtoBE(const AValue: DWord) : DWord
function NtoBE(const AValue: Int64) : Int64
function NtoBE(const AValue: QWord) : QWord
```

Visibility: default

Description: `NToBE` will rearrange the bytes in a natively-ordered number to the Big-Endian order. That is, for a Little-Endian processor, it will invert the order of the bytes and for a big-endian processor, it will do nothing.

See also: BEtoN (1213), LEtoN (1261), NtoLE (1267)

29.9.128 NtoLE

Synopsis: Convert Native-ordered integer to a Little Endian-ordered integer

Declaration:
```
function NtoLE(const AValue: SmallInt) : SmallInt
function NtoLE(const AValue: Word) : Word
function NtoLE(const AValue: LongInt) : LongInt
function NtoLE(const AValue: DWord) : DWord
function NtoLE(const AValue: Int64) : Int64
function NtoLE(const AValue: QWord) : QWord
```

Visibility: default

Description: `NToLE` will rearrange the bytes in a natively-ordered number to the little-Endian order. That is, for a Big-Endian processor, it will invert the order of the bytes and for a Little-Endian processor, it will do nothing.

See also: BEtoN (1213), LEtoN (1261), NtoBE (1266)

29.9.129 Null

Synopsis: Null variant

Declaration:
```
function Null : Variant
```

Visibility: default

29.9.130 OctStr

Synopsis: Convert integer to a string with octal representation.

Declaration:
```
function OctStr(Val: LongInt;cnt: Byte) : shortstring
function OctStr(Val: Int64;cnt: Byte) : shortstring
function OctStr(Val: qword;cnt: Byte) : shortstring
```

Visibility: default

Description: `OctStr` returns a string with the octal representation of `Value`. The string has exactly `cnt` charaters.

Errors: None.

See also: Str (1318), Val (1331), BinStr (1213), HexStr (1247)

Listing: ./refex/ex112.pp

```
Program example112;

{ Program to demonstrate the OctStr function }

Const Value = 45678;

Var I : longint;

begin
  For I:=1 to 10 do
    WriteIn (OctStr(Value,I));
  For I:=1 to 16 do
    WriteIn (OctStr(I,3));
end.
```

29.9.131 odd

Synopsis: Is a value odd or even ?

Declaration:
```
function odd(l: LongInt) : Boolean
function odd(l: LongWord) : Boolean
function odd(l: Int64) : Boolean
function odd(l: QWord) : Boolean
```

Visibility: default

Description: `Odd` returns `True` if `X` is odd, or `False` otherwise.

Errors: None.

See also: Abs (1205), Ord (1292)

Listing: ./refex/ex43.pp

```
Program Example43;

{ Program to demonstrate the Odd function. }

begin
  If Odd(1) Then
    WriteIn ('Everything OK with 1 !');
  If Not Odd(2) Then
    WriteIn ('Everything OK with 2 !');
end.
```

29.9.132 Ofs

Synopsis: Return offset of a variable.

Declaration:
```
function Ofs(var X) : LongInt
```

Visibility: default

Description: `Ofs` returns the offset of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always the complete address of the variable, since Free Pascal is a 32 bit compiler.

Errors: None.

See also: DSeg (1228), CSeg (1225), Seg (1309), Ptr (1296)

Listing: ./refex/ex44.pp

```
Program Example44;

{ Program to demonstrate the Ofs function. }

Var W : Pointer;


begin
  W:=Pointer(Ofs(W)); { W contains its own offset. }
end.
```

29.9.133 operator *(variant, variant): variant

Synopsis:

Declaration:
```
function operator *(variant, variant): variant(const op1: variant;
                                                const op2: variant)
                                                 : variant
```

Visibility: default

Description:

Errors:

29.9.134 operator **(variant, variant): variant

Synopsis:

Declaration:
```
function operator **(variant, variant): variant(const op1: variant;
                                                 const op2: variant)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.135 operator +(variant, variant): variant

Synopsis:

Declaration:
```
function operator +(variant, variant): variant(const op1: variant;
                                                const op2: variant)
                                                 : variant
```

Visibility: default

Description:

Errors:

29.9.136 operator -(variant): variant

Synopsis:

Declaration:
```
function operator -(variant): variant(const op: variant) : variant
```

Visibility: default

Description:

Errors:

29.9.137 operator -(variant, variant): variant

Synopsis:

Declaration:
```
function operator -(variant, variant): variant(const op1: variant;
                                                const op2: variant)
                                                 : variant
```

Visibility: default

Description:

Errors:

29.9.138 operator /(variant, variant): variant

Synopsis:

Declaration:
```
function operator /(variant, variant): variant(const op1: variant;
                                                const op2: variant)
                                                 : variant
```

Visibility: default

Description:

Errors:

29.9.139 operator :=(ansistring): olevariant

Synopsis:

Declaration:
```
function operator :=(ansistring): olevariant(const source: ansistring)
                                              : olevariant
```

Visibility: default

Description:

Errors:

29.9.140 operator :=(ansistring): variant

Synopsis:

Declaration:
```
function operator :=(ansistring): variant(const source: ansistring)
                                           : variant
```

Visibility: default

Description:

Errors:

29.9.141 operator :=(Boolean): olevariant

Synopsis:

Declaration:
```
function operator :=(Boolean): olevariant(const source: Boolean)
                                                  : olevariant
```

Visibility: default

Description:

Errors:

29.9.142 operator :=(Boolean): variant

Synopsis:

Declaration:
```
function operator :=(Boolean): variant(const source: Boolean) : variant
```

Visibility: default

Description:

Errors:

29.9.143 operator :=(Byte): olevariant

Synopsis:

Declaration:
```
function operator :=(Byte): olevariant(const source: Byte) : olevariant
```

Visibility: default

Description:

Errors:

29.9.144 operator :=(Byte): variant

Synopsis:

Declaration:
```
function operator :=(Byte): variant(const source: Byte) : variant
```

Visibility: default

Description:

Errors:

29.9.145 operator :=(Char): olevariant

Synopsis:

Declaration:
```
function operator :=(Char): olevariant(const source: Char) : olevariant
```

Visibility: default

Description:

Errors:

29.9.146 operator :=(Char): variant

Synopsis:

Declaration:
```
function operator :=(Char): variant(const source: Char) : variant
```

Visibility: default

Description:

Errors:

29.9.147 operator :=(currency): olevariant

Synopsis:

Declaration:
```
function operator :=(currency): olevariant(const source: currency)
                                                    : olevariant
```

Visibility: default

Description:

Errors:

29.9.148 operator :=(currency): variant

Synopsis:

Declaration:
```
function operator :=(currency): variant(const source: currency)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.149 operator :=(double): olevariant

Synopsis:

Declaration:
```
function operator :=(double): olevariant(const source: double)
                                                  : olevariant
```

Visibility: default

Description:

Errors:

29.9.150 operator :=(double): variant

Synopsis:

Declaration:
```
function operator :=(double): variant(const source: double) : variant
```

Visibility: default

Description:

Errors:

29.9.151 operator :=(DWord): olevariant

Synopsis:

Declaration:
```
function operator :=(DWord): olevariant(const source: DWord)
                                        : olevariant
```

Visibility: default

Description:

Errors:

29.9.152 operator :=(DWord): variant

Synopsis:

Declaration:
```
function operator :=(DWord): variant(const source: DWord) : variant
```

Visibility: default

Description:

Errors:

29.9.153 operator :=(Int64): olevariant

Synopsis:

Declaration:
```
function operator :=(Int64): olevariant(const source: Int64)
                                        : olevariant
```

Visibility: default

Description:

Errors:

29.9.154 operator :=(Int64): variant

Synopsis:

Declaration:
```
function operator :=(Int64): variant(const source: Int64) : variant
```

Visibility: default

Description:

Errors:

29.9.155 operator :=(longbool): olevariant

Synopsis:

Declaration:
```
function operator :=(longbool): olevariant(const source: longbool)
                                           : olevariant
```

Visibility: default

Description:

Errors:

29.9.156 operator :=(longbool): variant

Synopsis:

Declaration:
```
function operator :=(longbool): variant(const source: longbool)
                                                    : variant
```

Visibility: default

Description:

Errors:

29.9.157 operator :=(LongInt): olevariant

Synopsis:

Declaration:
```
function operator :=(LongInt): olevariant(const source: LongInt)
                                                      : olevariant
```

Visibility: default

Description:

Errors:

29.9.158 operator :=(LongInt): variant

Synopsis:

Declaration:
```
function operator :=(LongInt): variant(const source: LongInt) : variant
```

Visibility: default

Description:

Errors:

29.9.159 operator :=(olevariant): ansistring

Synopsis:

Declaration:
```
function operator :=(olevariant): ansistring(const source: olevariant)
                                                        : ansistring
```

Visibility: default

Description:

Errors:

29.9.160 operator :=(olevariant): Boolean

Synopsis:

Declaration:
```
function operator :=(olevariant): Boolean(const source: olevariant)
                                                        : Boolean
```

Visibility: default

Description:

Errors:

29.9.161 operator :=(olevariant): Byte

Synopsis:

Declaration:
```
function operator :=(olevariant): Byte(const source: olevariant) : Byte
```

Visibility: default

Description:

Errors:

29.9.162 operator :=(olevariant): Char

Synopsis:

Declaration:
```
function operator :=(olevariant): Char(const source: olevariant) : Char
```

Visibility: default

Description:

Errors:

29.9.163 operator :=(olevariant): currency

Synopsis:

Declaration:
```
function operator :=(olevariant): currency(const source: olevariant)
                                                      : currency
```

Visibility: default

Description:

Errors:

29.9.164 operator :=(olevariant): double

Synopsis:

Declaration:
```
function operator :=(olevariant): double(const source: olevariant)
                                                     : double
```

Visibility: default

Description:

Errors:

29.9.165 operator :=(olevariant): DWord

Synopsis:

Declaration:
```
function operator :=(olevariant): DWord(const source: olevariant)
                                                    : DWord
```

Visibility: default

Description:

Errors:

29.9.166 operator :=(olevariant): Int64

Synopsis:

Declaration:
```
function operator :=(olevariant): Int64(const source: olevariant)
                                                    : Int64
```

Visibility: default

Description:

Errors:

29.9.167 operator :=(olevariant): longbool

Synopsis:

Declaration:
```
function operator :=(olevariant): longbool(const source: olevariant)
                                                      : longbool
```

Visibility: default

Description:

Errors:

29.9.168 operator :=(olevariant): LongInt

Synopsis:

Declaration:
```
function operator :=(olevariant): LongInt(const source: olevariant)
                                                     : LongInt
```

Visibility: default

Description:

Errors:

29.9.169 operator :=(olevariant): qword

Synopsis:

Declaration:
```
function operator :=(olevariant): qword(const source: olevariant)
                                                    : qword
```

Visibility: default

Description:

Errors:

29.9.170 operator :=(olevariant): Real

Declaration:
```
function operator :=(olevariant): Real(const source: olevariant) : Real
```

Visibility: default

29.9.171 operator :=(olevariant): ShortInt

Synopsis:

Declaration:
```
function operator :=(olevariant): ShortInt(const source: olevariant)
                                                  : ShortInt
```

Visibility: default

Description:

Errors:

29.9.172 operator :=(olevariant): shortstring

Synopsis:

Declaration:
```
function operator :=(olevariant): shortstring(const source: olevariant)
                                                      : shortstring
```

Visibility: default

Description:

Errors:

29.9.173 operator :=(olevariant): SmallInt

Synopsis:

Declaration:
```
function operator :=(olevariant): SmallInt(const source: olevariant)
                                                  : SmallInt
```

Visibility: default

Description:

Errors:

29.9.174 operator :=(olevariant): TDateTime

Synopsis:

Declaration:
```
function operator :=(olevariant): TDateTime(const source: olevariant)
                                                        : TDateTime
```

Visibility: default

Description:

Errors:

29.9.175 operator :=(olevariant): TError

Synopsis:

Declaration:
```
function operator :=(olevariant): TError(const source: olevariant)
                                                    : TError
```

Visibility: default

Description:

Errors:

29.9.176 operator :=(olevariant): variant

Synopsis:

Declaration:
```
function operator :=(olevariant): variant(const source: olevariant)
                                                     : variant
```

Visibility: default

Description:

Errors:

29.9.177 operator :=(olevariant): widechar

Synopsis:

Declaration:
```
function operator :=(olevariant): widechar(const source: olevariant)
                                                      : widechar
```

Visibility: default

Description:

Errors:

29.9.178 operator :=(olevariant): widestring

Synopsis:

Declaration:
```
function operator :=(olevariant): widestring(const source: olevariant)
                                                    : widestring
```

Visibility: default

Description:

Errors:

29.9.179 operator :=(olevariant): Word

Synopsis:

Declaration:
```
function operator :=(olevariant): Word(const source: olevariant) : Word
```

Visibility: default

Description:

Errors:

29.9.180 operator :=(olevariant): wordbool

Synopsis:

Declaration:
```
function operator :=(olevariant): wordbool(const source: olevariant)
                                                  : wordbool
```

Visibility: default

Description:

Errors:

29.9.181 operator :=(qword): olevariant

Synopsis:

Declaration:
```
function operator :=(qword): olevariant(const source: qword)
                                               : olevariant
```

Visibility: default

Description:

Errors:

29.9.182 operator :=(qword): variant

Synopsis:

Declaration:
```
function operator :=(qword): variant(const source: qword) : variant
```

Visibility: default

Description:

Errors:

29.9.183 operator :=(Real): olevariant

Declaration: `function operator :=(Real): olevariant(const source: Real) : olevariant`

Visibility: default

29.9.184 operator :=(Real): variant

Declaration: `function operator :=(Real): variant(const source: Real) : variant`

Visibility: default

29.9.185 operator :=(real48): double

Synopsis:

Declaration: `function operator :=(real48): double(b: real48) : double`

Visibility: default

Description:

Errors:

29.9.186 operator :=(ShortInt): olevariant

Synopsis:

Declaration:
```
function operator :=(ShortInt): olevariant(const source: ShortInt)
                                                    : olevariant
```

Visibility: default

Description:

Errors:

29.9.187 operator :=(ShortInt): variant

Synopsis:

Declaration:
```
function operator :=(ShortInt): variant(const source: ShortInt)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.188 operator :=(shortstring): olevariant

Synopsis:

Declaration:
```
function operator :=(shortstring): olevariant(const source: shortstring)
                                                : olevariant
```

Visibility: default

Description:

Errors:

29.9.189 operator :=(shortstring): variant

Synopsis:

Declaration:
```
function operator :=(shortstring): variant(const source: shortstring)
                                             : variant
```

Visibility: default

Description:

Errors:

29.9.190 operator :=(SmallInt): olevariant

Synopsis:

Declaration:
```
function operator :=(SmallInt): olevariant(const source: SmallInt)
                                             : olevariant
```

Visibility: default

Description:

Errors:

29.9.191 operator :=(SmallInt): variant

Synopsis:

Declaration:
```
function operator :=(SmallInt): variant(const source: SmallInt)
                                          : variant
```

Visibility: default

Description:

Errors:

29.9.192 operator :=(TDateTime): olevariant

Synopsis:

Declaration:
```
function operator :=(TDateTime): olevariant(const source: TDateTime)
                                                        : olevariant
```

Visibility: default

Description:

Errors:

29.9.193 operator :=(TDateTime): variant

Synopsis:

Declaration:
```
function operator :=(TDateTime): variant(const source: TDateTime)
                                                    : variant
```

Visibility: default

Description:

Errors:

29.9.194 operator :=(TError): olevariant

Synopsis:

Declaration:
```
function operator :=(TError): olevariant(const source: TError)
                                                    : olevariant
```

Visibility: default

Description:

Errors:

29.9.195 operator :=(TError): variant

Synopsis:

Declaration:
```
function operator :=(TError): variant(const source: TError) : variant
```

Visibility: default

Description:

Errors:

29.9.196 operator :=(variant): ansistring

Synopsis:

Declaration:
```
function operator :=(variant): ansistring(const source: variant)
                                                  : ansistring
```

Visibility: default

Description:

Errors:

29.9.197 operator :=(variant): Boolean

Synopsis:

Declaration:
```
function operator :=(variant): Boolean(const source: variant) : Boolean
```

Visibility: default

Description:

Errors:

29.9.198 operator :=(variant): Byte

Synopsis:

Declaration:
```
function operator :=(variant): Byte(const source: variant) : Byte
```

Visibility: default

Description:

Errors:

29.9.199 operator :=(variant): Char

Synopsis:

Declaration:
```
function operator :=(variant): Char(const source: variant) : Char
```

Visibility: default

Description:

Errors:

29.9.200 operator :=(variant): currency

Synopsis:

Declaration:
```
function operator :=(variant): currency(const source: variant)
                                               : currency
```

Visibility: default

Description:

Errors:

29.9.201 operator :=(variant): double

Synopsis:

Declaration: `function operator :=(variant): double(const source: variant) : double`

Visibility: default

Description:

Errors:

29.9.202 operator :=(variant): DWord

Synopsis:

Declaration: `function operator :=(variant): DWord(const source: variant) : DWord`

Visibility: default

Description:

Errors:

29.9.203 operator :=(variant): Int64

Synopsis:

Declaration: `function operator :=(variant): Int64(const source: variant) : Int64`

Visibility: default

Description:

Errors:

29.9.204 operator :=(variant): longbool

Synopsis:

Declaration:
```
function operator :=(variant): longbool(const source: variant)
                                        : longbool
```

Visibility: default

Description:

Errors:

29.9.205 operator :=(variant): LongInt

Synopsis:

Declaration: `function operator :=(variant): LongInt(const source: variant) : LongInt`

Visibility: default

Description:

Errors:

29.9.206 operator :=(variant): olevariant

Synopsis:

Declaration:
```
function operator :=(variant): olevariant(const source: variant)
                                                    : olevariant
```

Visibility: default

Description:

Errors:

29.9.207 operator :=(variant): qword

Synopsis:

Declaration:
```
function operator :=(variant): qword(const source: variant) : qword
```

Visibility: default

Description:

Errors:

29.9.208 operator :=(variant): Real

Declaration:
```
function operator :=(variant): Real(const source: variant) : Real
```

Visibility: default

29.9.209 operator :=(variant): ShortInt

Synopsis:

Declaration:
```
function operator :=(variant): ShortInt(const source: variant)
                                                  : ShortInt
```

Visibility: default

Description:

Errors:

29.9.210 operator :=(variant): shortstring

Synopsis:

Declaration:
```
function operator :=(variant): shortstring(const source: variant)
                                                    : shortstring
```

Visibility: default

Description:

Errors:

29.9.211 operator :=(variant): SmallInt

Synopsis:

Declaration:
```
function operator :=(variant): SmallInt(const source: variant)
                                        : SmallInt
```

Visibility: default

Description:

Errors:

29.9.212 operator :=(variant): TDateTime

Synopsis:

Declaration:
```
function operator :=(variant): TDateTime(const source: variant)
                                         : TDateTime
```

Visibility: default

Description:

Errors:

29.9.213 operator :=(variant): TError

Synopsis:

Declaration:
```
function operator :=(variant): TError(const source: variant) : TError
```

Visibility: default

Description:

Errors:

29.9.214 operator :=(variant): widechar

Synopsis:

Declaration:
```
function operator :=(variant): widechar(const source: variant)
                                        : widechar
```

Visibility: default

Description:

Errors:

29.9.215 operator :=(variant): widestring

Synopsis:

Declaration:
```
function operator :=(variant): widestring(const source: variant)
                                                    : widestring
```

Visibility: default

Description:

Errors:

29.9.216 operator :=(variant): Word

Synopsis:

Declaration:
```
function operator :=(variant): Word(const source: variant) : Word
```

Visibility: default

Description:

Errors:

29.9.217 operator :=(variant): wordbool

Synopsis:

Declaration:
```
function operator :=(variant): wordbool(const source: variant)
                                                  : wordbool
```

Visibility: default

Description:

Errors:

29.9.218 operator :=(widechar): olevariant

Synopsis:

Declaration:
```
function operator :=(widechar): olevariant(const source: widechar)
                                                     : olevariant
```

Visibility: default

Description:

Errors:

29.9.219 operator :=(widechar): variant

Synopsis:

Declaration:
```
function operator :=(widechar): variant(const source: widechar)
                                                    : variant
```

Visibility: default

Description:

Errors:

29.9.220 operator :=(widestring): olevariant

Synopsis:

Declaration:
```
function operator :=(widestring): olevariant(const source: widestring)
                                                        : olevariant
```

Visibility: default

Description:

Errors:

29.9.221 operator :=(widestring): variant

Synopsis:

Declaration:
```
function operator :=(widestring): variant(const source: widestring)
                                                    : variant
```

Visibility: default

Description:

Errors:

29.9.222 operator :=(Word): olevariant

Synopsis:

Declaration:
```
function operator :=(Word): olevariant(const source: Word) : olevariant
```

Visibility: default

Description:

Errors:

29.9.223 operator :=(Word): variant

Synopsis:

Declaration:
```
function operator :=(Word): variant(const source: Word) : variant
```

Visibility: default

Description:

Errors:

29.9.224 operator :=(wordbool): olevariant

Synopsis:

Declaration:
```
function operator :=(wordbool): olevariant(const source: wordbool)
                                                    : olevariant
```

Visibility: default

Description:

Errors:

29.9.225 operator :=(wordbool): variant

Synopsis:

Declaration:
```
function operator :=(wordbool): variant(const source: wordbool)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.226 operator <(variant, variant): Boolean

Synopsis:

Declaration:
```
function operator <(variant, variant): Boolean(const op1: variant;
                                                 const op2: variant)
                                                  : Boolean
```

Visibility: default

Description:

Errors:

29.9.227 operator <=(variant, variant): Boolean

Synopsis:

Declaration:
```
function operator <=(variant, variant): Boolean(const op1: variant;
                                                  const op2: variant)
                                                   : Boolean
```

Visibility: default

Description:

Errors:

29.9.228 operator =(variant, variant): Boolean

Synopsis:

Declaration:
```
function operator =(variant, variant): Boolean(const op1: variant;
                                               const op2: variant)
                                                : Boolean
```

Visibility: default

Description:

Errors:

29.9.229 operator >(variant, variant): Boolean

Synopsis:

Declaration:
```
function operator >(variant, variant): Boolean(const op1: variant;
                                               const op2: variant)
                                                : Boolean
```

Visibility: default

Description:

Errors:

29.9.230 operator >=(variant, variant): Boolean

Synopsis:

Declaration:
```
function operator >=(variant, variant): Boolean(const op1: variant;
                                                const op2: variant)
                                                 : Boolean
```

Visibility: default

Description:

Errors:

29.9.231 operator and(variant, variant): variant

Synopsis:

Declaration:
```
function operator and(variant, variant): variant(const op1: variant;
                                                 const op2: variant)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.232 operator div(variant, variant): variant

Synopsis:

Declaration:
```
function operator div(variant, variant): variant(const op1: variant;
                                                  const op2: variant)
                                                   : variant
```

Visibility: default

Description:

Errors:

29.9.233 operator mod(variant, variant): variant

Synopsis:

Declaration:
```
function operator mod(variant, variant): variant(const op1: variant;
                                                  const op2: variant)
                                                   : variant
```

Visibility: default

Description:

Errors:

29.9.234 operator not(variant): variant

Synopsis:

Declaration:
```
function operator not(variant): variant(const op: variant) : variant
```

Visibility: default

Description:

Errors:

29.9.235 operator or(variant, variant): variant

Synopsis:

Declaration:
```
function operator or(variant, variant): variant(const op1: variant;
                                                 const op2: variant)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.236 operator shl(variant, variant): variant

Synopsis:

Declaration:
```
function operator shl(variant, variant): variant(const op1: variant;
                                                 const op2: variant)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.237 operator shr(variant, variant): variant

Synopsis:

Declaration:
```
function operator shr(variant, variant): variant(const op1: variant;
                                                 const op2: variant)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.238 operator xor(variant, variant): variant

Synopsis:

Declaration:
```
function operator xor(variant, variant): variant(const op1: variant;
                                                 const op2: variant)
                                                  : variant
```

Visibility: default

Description:

Errors:

29.9.239 Ord

Synopsis: Return ordinal value of an ordinal type.

Declaration:
```
function Ord(X: TOrdinal) : LongInt
```

Visibility: default

Description: `Ord` returns the Ordinal value of a ordinal-type variable `X`.

Historical note:

Originally, Pascal did not have typecasts and ord was a necessary function in order to do certain operations on non-integer ordinal types. With the arrival of typecasting a generic approach became possible, making ord mostly obselete. However ord is not considered deprecated and remains in wide use today.

Errors: None.

See also: Chr (1216), Succ (1321), Pred (1296), High (1249), Low (1263)

Listing: ./refex/ex45.pp

```
Program Example45;

{ Program to demonstrate the Ord,Pred,Succ functions. }

Type
  TEnum = (Zero, One, Two, Three, Four);

Var
  X : Longint;
  Y : TEnum;

begin
  X:=125;
  Writeln (Ord(X));  { Prints 125 }
  X:=Pred(X);
  Writeln (Ord(X));  { prints 124 }
  Y:= One;
  Writeln (Ord(y));  { Prints 1 }
  Y:=Succ(Y);
  Writeln (Ord(Y));  { Prints 2}
end.
```

29.9.240 Paramcount

Synopsis: Return number of command-line parameters passed to the program.

Declaration: `function Paramcount : LongInt`

Visibility: default

Description: `Paramcount` returns the number of command-line arguments. If no arguments were given to the running program, `0` is returned.

Errors: None.

See also: Paramstr (1294)

Listing: ./refex/ex46.pp

```
Program Example46;

{ Program to demonstrate the ParamCount and ParamStr functions. }
Var
  I : Longint;

begin
  Writeln (paramstr(0),' : Got ',ParamCount,' command-line parameters: ');
  For i:=1 to ParamCount do
    Writeln (ParamStr (i));
end.
```

29.9.241 ParamStr

Synopsis: Return value of a command-line argument.

Declaration: `function ParamStr(l: LongInt) : String`

Visibility: default

Description: `Paramstr` returns the `L`-th command-line argument. `L` must be between `0` and `Paramcount`, these values included. The zeroth argument is the path and file name with which the program was started.

The command-line parameters will be truncated to a length of 255, even though the operating system may support bigger command-lines. The `Objpas` unit (used in `objfpc` or `delphi` mode) defines versions of `Paramstr` which return the full-length command-line arguments, using ansistrings.

In the interest of portability, the `ParamStr` function tries to behave the same on all operating systems: like the original `ParamStr` function in Turbo Pascal. This means even on Unix, paramstr(0) returns the full path to the program executable. A notable exception is Mac OS X, where the return value depends on how the application was started. It may be that just the name of the application is returned (in case of a command-line launch)

In general, it's a bad idea to rely on the location of the binary. Often, this goes against best OS practices. Configuration data should (or can) not be stored next to the binary, but on designated locations. What locations these are, is very much operating system dependent. Therefore, `ParamStr(0)` should be used with care.

For an example, see Paramcount (1293).

Errors: None.

See also: Paramcount (1293)

29.9.242 pi

Synopsis: Return the value of PI.

Declaration: `function pi : ValReal`

Visibility: default

Description: `Pi` returns the value of Pi (3.1415926535897932385).

Errors: None.

See also: Cos (1224), Sin (1315)

Listing: ./refex/ex47.pp

```
Program Example47;

{ Program to demonstrate the Pi function. }

begin
  Writeln (Pi);          {3.1415926}
  Writeln (Sin(Pi));
end.
```

29.9.243 Pos

Synopsis: Search for substring in a string.

Declaration:
```
function Pos(const substr: shortstring;const s: shortstring) : SizeInt
function Pos(C: Char;const s: shortstring) : SizeInt
function Pos(const Substr: ShortString;const Source: AnsiString)
            : SizeInt
function pos(const substr: shortstring;c: Char) : SizeInt
function Pos(const Substr: AnsiString;const Source: AnsiString)
            : SizeInt
function Pos(c: Char;const s: AnsiString) : SizeInt
function Pos(const Substr: WideString;const Source: WideString)
            : SizeInt
function Pos(c: Char;const s: WideString) : SizeInt
function Pos(c: WideChar;const s: WideString) : SizeInt
function Pos(c: WideChar;const s: AnsiString) : SizeInt
function Pos(c: AnsiString;const s: WideString) : SizeInt
function Pos(c: WideString;const s: AnsiString) : SizeInt
function Pos(c: ShortString;const s: WideString) : SizeInt
function Pos(c: Char;const v: Variant) : SizeInt
function Pos(s: ShortString;const v: Variant) : SizeInt
function Pos(a: AnsiString;const v: Variant) : SizeInt
function Pos(w: WideString;const v: Variant) : SizeInt
function Pos(v: Variant;const c: Char) : SizeInt
function Pos(v: Variant;const s: ShortString) : SizeInt
function Pos(v: Variant;const a: AnsiString) : SizeInt
function Pos(v: Variant;const w: WideString) : SizeInt
function Pos(v1: Variant;const v2: Variant) : SizeInt
```

Visibility: default

Description: `Pos` returns the index of `Substr` in `S`, if `S` contains `Substr`. In case `Substr` isn't found, `0` is returned. The search is case-sensitive.

Errors: None

See also: Length (1260), Copy (1224), Delete (1227), Insert (1255)

Listing: ./refex/ex48.pp

```
Program Example48;

{ Program to demonstrate the Pos function. }

Var
  S : String;

begin
  S:='The first space in this sentence is at position : ';
  Writeln (S,pos(' ',S));
  S:='The last letter of the alphabet doesn''t appear in this sentence ';
  If (Pos ('Z',S)=0) and (Pos('z',S)=0) then
    Writeln (S);
end.
```

29.9.244 Pred

Synopsis: Return previous element for an ordinal type.

Declaration: `function Pred(X: TOrdinal) : TOrdinal`

Visibility: default

Description: `Pred` returns the element that precedes the element that was passed to it. If it is applied to the first value of the ordinal type, and the program was compiled with range checking on (\var{\{\$R+\}}, then a run-time error will be generated.

for an example, see Ord (1292)

Errors: Run-time error 201 is generated when the result is out of range.

See also: Ord (1292), Pred (1296), High (1249), Low (1263)

29.9.245 prefetch

Synopsis: Prefetch a memory location

Declaration: `procedure prefetch(const mem)`

Visibility: default

Description: `Prefetch` can be used to optimize the CPU behaviour by already loading a memory location. It is mainly used as a hint for those processors that support it.

Errors: None.

29.9.246 ptr

Synopsis: Combine segment and offset to pointer

Declaration: `function ptr(sel: LongInt;off: LongInt) : farpointer`

Visibility: default

Description: `Ptr` returns a pointer, pointing to the address specified by segment `Sel` and offset `Off`.

Remark:

1.In the 32-bit flat-memory model supported by Free Pascal, this function is obsolete.

2.The returned address is simply the offset.

Errors: None.

See also: Addr (1206)

Listing: ./refex/ex59.pp

```
Program Example59;

{ Program to demonstrate the Ptr (compability) function.
}

type pString = ^String;

Var P : pString;
    S : String;
```

```
begin
  S:='Hello, World !';
  P:=pString(Ptr(Seg(S),Longint(Ofs(S))));
  {P now points to S !}
  Writeln (P^);
end.
```

29.9.247 RaiseList

Synopsis: List of currently raised exceptions.

Declaration:
```
function RaiseList : PExceptObject
```

Visibility: default

Description: `RaiseList` returns a pointer to the list of currently raised exceptions (i.e. a pointer to the first exception block.

Errors:

29.9.248 Random

Synopsis: Generate random number

Declaration:
```
function Random(l: LongInt) : LongInt
function Random(l: Int64) : Int64
function Random : extended
```

Visibility: default

Description: `Random` returns a random number larger or equal to `0` and strictly less than `L`. If the argument `L` is omitted, a Real number between 0 and 1 is returned. (0 included, 1 excluded)

Errors: None.

See also: Randomize (1298)

Listing: ./refex/ex49.pp

```
Program Example49;

{ Program to demonstrate the Random and Randomize functions. }

Var I,Count,guess : Longint;
    R : Real;

begin
  Randomize; { This way we generate a new sequence every time
               the program is run}
  Count:=0;
  For i:=1 to 1000 do
    If Random>0.5 then inc(Count);
  Writeln ('Generated ',Count,' numbers > 0.5');
  Writeln ('out of 1000 generated numbers.');
  count:=0;
  For i:=1 to 5 do
```

```
    begin
    write ('Guess a number between 1 and 5 : ');
    readln(Guess);
    If Guess=Random(5)+1 then inc(count);
    end;
  WriteIn ('You guessed ',Count,' out of 5 correct.');
end.
```

29.9.249 Randomize

Synopsis: Initialize random number generator

Declaration: `procedure Randomize`

Visibility: default

Description: `Randomize` initializes the random number generator of Free Pascal, by giving a value to `Randseed`, calculated with the system clock.

For an example, see Random (1297).

Errors: None.

See also: Random (1297)

29.9.250 Read

Synopsis: Read from a text file into variable

Declaration:
```
procedure Read(var F: Text;Args: Arguments)
procedure Read(Args: Arguments)
```

Visibility: default

Description: `Read` reads one or more values from a file `F`, and stores the result in `V1`, `V2`, etc.; If no file `F` is specified, then standard input is read. If `F` is of type `Text`, then the variables `V1, V2` etc. must be of type `Char`, `Integer`, `Real`, `String`. If `F` is a typed file, then each of the variables must be of the type specified in the declaration of `F`. Untyped files are not allowed as an argument.

In earlier versions of FPC, it was also allowed to read `Pchar` null-terminated strings, but this has been removed, since there is no buffer checking possible.

Errors: If no data is available, a run-time error is generated. This behavior can be controlled with the \var{\{\$i\}} compiler switch.

See also: Readln (1299), Blockread (1213), Write (1334), Blockwrite (1214)

Listing: ./refex/ex50.pp

```
Program Example50;

{ Program to demonstrate the Read(Ln) function. }

Var S : String;
    C : Char;
    F : File of char;

begin
```

```
  Assign (F,'ex50.pp');
  Reset (F);
  C:='A';
  Writeln ('The characters before the first space in ex50.pp are : ');
  While not Eof(f) and (C<>' ') do
    Begin
    Read (F,C);
    Write (C);
    end;
 Writeln;
 Close (F);
 Writeln ('Type some words. An empty line ends the program.');
 repeat
   Readln (S);
 until S='';
end.
```

29.9.251 ReadBarrier

Synopsis: Memory Read Barrier

Declaration: `procedure ReadBarrier`

Visibility: default

Description: `ReadBarrier` is a low-level instruction to force a read barrier in the CPU: all memory reads before the instruction will be finished before this instruction, before memory reads after the instruction occur.

See also: ReadDependencyBarrier (1299), ReadWriteBarrier (1300), WriteBarrier (1334)

29.9.252 ReadDependencyBarrier

Synopsis: Memory Read Dependency Barrier

Declaration: `procedure ReadDependencyBarrier`

Visibility: default

Description: `ReadDependencyBarrier` is a low-level instruction to force a read barrier in the CPU: all memory reads (loads) depending on previous loads are separate from the ones following the instruction.

See also: ReadBarrier (1299), ReadWriteBarrier (1300), WriteBarrier (1334)

29.9.253 ReadLn

Synopsis: Read from a text file into variable and goto next line

Declaration:
```
procedure ReadLn(var F: Text;Args: Arguments)
procedure ReadLn(Args: Arguments)
```

Visibility: default

Description: `Read` reads one or more values from a file `F`, and stores the result in `V1`, `V2`, etc. After that it goes to the next line in the file. The end of the line is marked by the `LineEnding` character sequence (which is platform dependent). The end-of-line marker is not considered part of the line and is ignored.

If no file `F` is specified, then standard input is read. The variables `V1, V2` etc. must be of type `Char`, `Integer`, `Real`, `String` or `PChar`.

For an example, see Read (1298).

Errors: If no data is available, a run-time error is generated. This behavior can be controlled with the \var{\{\$i\}} compiler switch.

See also: Read (1298), Blockread (1213), Write (1334), Blockwrite (1214)

29.9.254 ReadWriteBarrier

Synopsis: Memory read/write barrier

Declaration: `procedure ReadWriteBarrier`

Visibility: default

Description: `ReadWriteBarrier` is a low-level instruction to force a read/write barrier in the CPU: both read (Loads) and write (stores) operations before and after the barrier are separate.

See also: ReadBarrier (1299), ReadDependencyBarrier (1299), WriteBarrier (1334)

29.9.255 Real2Double

Synopsis: Convert Turbo Pascal style real to double.

Declaration: `function Real2Double(r: real48) : double`

Visibility: default

Description: The `Real2Double` function converts a Turbo Pascal style real (6 bytes long) to a native Free Pascal double type. It can be used e.g. to read old binary TP files with FPC and convert them to Free Pacal binary files.

Note that the assignment operator has been overloaded so a `Real48` type can be assigned directly to a double or extended.

Errors: None.

Listing: ./refex/ex110.pp

```
program Example110;

{ Program to demonstrate the Real2Double function. }

Var
  i : integer;
  R : Real48;
  D : Double;
  E : Extended;
  F : File of Real48;

begin
  Assign(F,'reals.dat');
  Reset(f);
  For I:=1 to 10 do
    begin
    Read(F,R);
```

```
    D:=Real2Double(R);
    Writeln('Real ',i,' : ',D);
    D:=R;
    Writeln('Real (direct to double)  ',i,' : ',D);
    E:=R;
    Writeln('Real (direct to Extended) ',i,' : ',E);
    end;
  Close(f);
end.
```

29.9.256 ReAllocMem

Synopsis: Re-allocate memory on the heap

Declaration: `function ReAllocMem(var p: pointer;Size: PtrInt) : pointer`

Visibility: default

Description: `ReAllocMem` resizes the memory pointed to by `P` so it has size `Size`. The value of `P` may change during this operation. The contents of the memory pointed to by `P` (if any) will be copied to the new location, but may be truncated if the newly allocated memory block is smaller in size. If a larger block is allocated, only the used memory is initialized, extra memory will not be zeroed out.

Note that `P` may be nil, in that case the behaviour of `ReAllocMem` is equivalent to `Getmem`.

See also: GetMem (1244), FreeMem (1242)

29.9.257 ReAllocMemory

Synopsis: Alias for ReAllocMem (1301)

Declaration: `function ReAllocMemory(var p: pointer;Size: PtrInt) : pointer`

Visibility: default

Description: `ReAllocMemory` is an alias for ReAllocMem (1301).

See also: ReAllocMem (1301)

29.9.258 ReleaseExceptionObject

Synopsis: Decrease the reference count of the current exception object.

Declaration: `procedure ReleaseExceptionObject`

Visibility: default

Description: `ReleaseExceptionObject` decreases the reference count of the current exception object. This should be called whenever a reference to the exception object was obtained via the AcquireExceptionObject (1206) call.

Calling this method is only valid within an except block.

Errors: If there is no current exception object, a run-time error 231 will occur.

See also: AcquireExceptionObject (1206)

29.9.259 Rename

Synopsis: Rename file on disk

Declaration:
```
procedure Rename(var f: File;const s: String)
procedure Rename(var f: File;p: PChar)
procedure Rename(var f: File;c: Char)
procedure Rename(var t: Text;const s: String)
procedure Rename(var t: Text;p: PChar)
procedure Rename(var t: Text;c: Char)
```

Visibility: default

Description: `Rename` changes the name of the assigned file `F` to `S`. `F` must be assigned, but not opened.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Erase (1232)

Listing: ./refex/ex77.pp

```
Program Example77;

{ Program to demonstrate the Rename function. }
Var F : Text;

begin
  Assign (F,paramstr(1));
  Rename (F,paramstr(2));
end.
```

29.9.260 Reset

Synopsis: Open file for reading

Declaration:
```
procedure Reset(var f: File;l: LongInt)
procedure Reset(var f: File)
procedure Reset(var f: TypedFile)
procedure Reset(var t: Text)
```

Visibility: default

Description: `Reset` opens a file `F` for reading. `F` can be any file type. If `F` is a text file, or refers to standard I/O (e.g : '') then it is opened read-only, otherwise it is opened using the mode specified in `filemode`. If `F` is an untyped file, the record size can be specified in the optional parameter `L`. A default value of 128 is used. File sharing is not taken into account when calling `Reset`.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Rewrite (1303), Assign (1210), Close (1217), Append (1208)

Listing: ./refex/ex51.pp

```
Program Example51;

{ Program to demonstrate the Reset function. }

Function FileExists (Name : String) : boolean;

Var F : File;

begin
  {$i-}
  Assign (F,Name);
  Reset (F);
  {$I+}
  FileExists:=(IoResult=0) and (Name<>'');
  Close (f);
end;

begin
  If FileExists (Paramstr(1)) then
    WriteIn ('File found')
  else
    WriteIn ('File NOT found');
end.
```

29.9.261 ResumeThread

Synopsis: Resume a suspended thread.

Declaration: `function ResumeThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `ResumeThread` causes a suspended thread (using SuspendThread (1321)) to resume it's execution. The thread is identified with it's handle or ID `threadHandle`.

The function returns zero if succesful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: SuspendThread (1321), KillThread (1260)

29.9.262 Rewrite

Synopsis: Open file for writing

Declaration:
```
procedure Rewrite(var f: File;l: LongInt)
procedure Rewrite(var f: File)
procedure Rewrite(var f: TypedFile)
procedure Rewrite(var t: Text)
```

Visibility: default

Description: `Rewrite` opens a file `F` for writing. `F` can be any file type. If `F` is an untyped or typed file, then it is opened for reading and writing. If `F` is an untyped file, the record size can be specified in the optional parameter `L`. Default a value of 128 is used. if `Rewrite` finds a file with the same name as `F`, this file is truncated to length `0`. If it doesn't find such a file, a new file is created. Contrary to Turbo Pascal, Free Pascal opens the file with mode `fmoutput`. If it should be opened in `fminout`

mode, an extra call to Reset (1302) is needed. File sharing is not taken into account when calling `Rewrite`.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Reset (1302), Assign (1210), Close (1217), Flush (1240), Append (1208)

Listing: ./refex/ex52.pp

```
Program Example52;

{ Program to demonstrate the Rewrite function. }

Var F : File;
    I : longint;

begin
  Assign (F,'Test.tmp');
  { Create the file. Recordsize is 4 }
  Rewrite (F,Sizeof(I));
  For I:=1 to 10 do
    BlockWrite (F,I,1);
  close (f);
  { F contains now a binary representation of
    10 longints going from 1 to 10 }
end.
```

29.9.263 rmdir

Synopsis: Remove directory when empty.

Declaration: `procedure rmdir(const s: String)`

Visibility: default

Description: `Rmdir` removes the directory `S`.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Chdir (1216), Mkdir (1264)

Listing: ./refex/ex53.pp

```
Program Example53;

{ Program to demonstrate the MkDir and RmDir functions. }

Const D : String[8] = 'TEST.DIR';

Var S : String;

begin
  Writeln ('Making directory ',D);
  Mkdir (D);
  Writeln ('Changing directory to ',D);
  ChDir (D);
```

```
  GetDir (0,S);
  WriteLn ('Current Directory is : ',S);
  WRiteln ('Going back');
  ChDir ('..');
  WriteLn ('Removing directory ',D);
  RmDir (D);
end.
```

29.9.264 round

Synopsis: Round floating point value to nearest integer number.

Declaration: `function round(d: ValReal) : Int64`

Visibility: default

Description: `Round` rounds `X` to the closest integer, which may be bigger or smaller than `X`.

In the case of `.5`, the algorithm uses "banker's rounding": `.5` values are always rounded towards the even number.

Errors: None.

See also: Frac (1241), Int (1256), Trunc (1327)

Listing: ./refex/ex54.pp

```
Program Example54;

{ Program to demonstrate the Round function. }

begin
  WriteLn (Round(1234.56));  { Prints 1235     }
  WriteLn (Round(-1234.56)); { Prints -1235    }
  WriteLn (Round(12.3456));  { Prints 12       }
  WriteLn (Round(-12.3456)); { Prints -12      }
  WriteLn (Round(2.5));      { Prints 2 (down) }
  WriteLn (Round(3.5));      { Prints 4 (up)   }

end.
```

29.9.265 RTLEventCreate

Synopsis: Create a new RTL event

Declaration: `function RTLEventCreate : PRTLEvent`

Visibility: default

Description: `RTLEventCreate` creates and initializes a new RTL event. RTL events are used to notify other threads that a certain condition is met, and to notify other threads of condition changes (conditional variables).

The function returns an initialized RTL event, which must be disposed of with RTLEventdestroy (1306)

`RTLEvent` is used mainly for the synchronize method.

See also: RTLEventDestroy (1306), RTLEventSet (1159), RTLEventReSet (1159), RTLEventWaitFor (1306)

29.9.266 RTLeventdestroy

Synopsis: Destroy a RTL Event

Declaration:
```
procedure RTLeventdestroy(state: PRTLEvent)
```

Visibility: default

Description: `RTLeventdestroy` destroys the RTL event `State`. After a call to `RTLeventdestroy`, the `State` RTL event may no longer be used.

See also: RTLEventCreate (1305), RTLEventReset (1159), RTLEventSet (1159)

29.9.267 RTLeventResetEvent

Synopsis: Reset an event

Declaration:
```
procedure RTLeventResetEvent(state: PRTLEvent)
```

Visibility: default

Description: `RTLeventResetEvent` resets the event: this should be used to undo the signaled state of an event. Resetting an event that is not set (or was already reset) has no effect.

See also: RTLEventCreate (1305), RTLEventDestroy (1306), RTLEventSetEvent (1306), RTLEventWaitFor (1306)

29.9.268 RTLeventSetEvent

Synopsis: Notify threads of the event.

Declaration:
```
procedure RTLeventSetEvent(state: PRTLEvent)
```

Visibility: default

Description: `RTLeventSetEvent` notifies other threads which are listening, that the event has occurred.

See also: RTLEventCreate (1305), RTLEventResetEvent (1306), RTLEventDestroy (1306), RTLEventWaitFor (1306)

29.9.269 RTLeventsync

Synopsis: Obsolete. Don't use

Declaration:
```
procedure RTLeventsync(m: trtlmethod;p: TProcedure)
```

Visibility: default

Description: `RTLeventsync` is obsolete, don't use it.

29.9.270 RTLeventWaitFor

Synopsis: Wait for an event.

Declaration:
```
procedure RTLeventWaitFor(state: PRTLEvent)
procedure RTLeventWaitFor(state: PRTLEvent;timeout: LongInt)
```

Visibility: default

Description: `RTLeventWaitFor` suspends the thread till the event occurs. The event will occur when another thread calls RTLEventSetEvent (1306) on `State`.

By default, the thread will be suspended indefinitely. However, if `TimeOut` is specified, then the thread will resume after timeout milliseconds have elapsed.

See also: RTLEventCreate (1305), RTLEventDestroy (1306), RTLEventSetEvent (1306), RTLEventWaitFor (1306)

29.9.271 RunError

Synopsis: Generate a run-time error.

Declaration:
```
procedure RunError(w: Word)
procedure RunError
```

Visibility: default

Description: `Runerror` stops the execution of the program, and generates a run-time error `ErrorCode`.

Errors: None.

See also: Exit (1234), Halt (1247)

Listing: ./refex/ex55.pp

```
Program Example55;

{ Program to demonstrate the RunError function. }

begin
  { The program will stop end emit a run-error 106 }
  RunError (106);
end.
```

29.9.272 Seek

Synopsis: Set file position

Declaration:
```
procedure Seek(var f: File;Pos: Int64)
```

Visibility: default

Description: `Seek` sets the file-pointer for file `F` to record Nr. `Count`. The first record in a file has `Count=0`. F can be any file type, except `Text`. If `F` is an untyped file, with no record size specified in Reset (1302) or Rewrite (1303), 128 is assumed.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Eof (1231), SeekEof (1308), SeekEoln (1309)

Listing: ./refex/ex56.pp

```
Program Example56;

{ Program to demonstrate the Seek function. }

Var
  F : File;
  I,j : longint;

begin
  { Create a file and fill it with data }
  Assign (F,'test.tmp');
  Rewrite(F); { Create file }
  Close(f);
  FileMode:=2;
  ReSet (F,Sizeof(i)); { Opened read/write }
  For I:=0 to 10 do
    BlockWrite (F,I,1);
  { Go Back to the begining of the file }
  Seek(F,0);
  For I:=0 to 10 do
    begin
    BlockRead (F,J,1);
    If J<>I then
      WriteIn ('Error: expected ' ,i,', got ',j);
    end;
  Close (f);
end.
```

29.9.273 SeekEOF

Synopsis: Set file position to end of file

Declaration:
```
function SeekEOF(var t: Text) : Boolean
function SeekEOF : Boolean
```

Visibility: default

Description: `SeekEof` returns `True` is the file-pointer is at the end of the file. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-file marker is reached.

If the end-of-file marker is reached, `True` is returned. Otherwise, False is returned.

If the parameter `F` is omitted, standard `Input` is assumed.

Remark: The SeekEOF function can only be used on real textfiles: when assigning the file to other kinds of (virtual) text files, the function may fail, although it will perform a number of tests to guard against wrong usage.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: Eof (1231), SeekEoln (1309), Seek (1307)

Listing: ./refex/ex57.pp

```
Program Example57;

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

```
Var C : Char;

begin
  { this will print all characters from standard input except
    Whitespace characters. }
  While Not SeekEof do
    begin
    Read (C);
    Write (C);
    end;
end.
```

29.9.274 SeekEOLn

Synopsis: Set file position to end of line

Declaration:
```
function SeekEOLn(var t: Text) : Boolean
function SeekEOLn : Boolean
```

Visibility: default

Description: `SeekEoln` returns `True` is the file-pointer is at the end of the current line. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-line marker is reached. If the end-of-line marker is reached, `True` is returned. Otherwise, False is returned. The end-of-line marker is defined as `#10`, the LineFeed character. If the parameter `F` is omitted, standard `Input` is assumed.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: Eof (1231), SeekEof (1308), Seek (1307)

Listing: ./refex/ex58.pp

```
Program Example58;

{ Program to demonstrate the SeekEoln function. }
Var
  C : Char;

begin
  { This will read the first line of standard output and print
    all characters except whitespace. }
  While not SeekEoln do
    Begin
    Read (c);
    Write (c);
    end;
end.
```

29.9.275 Seg

Synopsis: Return segment

Declaration:
```
function Seg(var X) : LongInt
```

Visibility: default

Description: `Seg` returns the segment of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always 0, since Free Pascal uses a flat 32/64 bit memory model. In such a memory model segments have no meaning.

Errors: None.

See also: DSeg (1228), CSeg (1225), Ofs (1268), Ptr (1296)

Listing: ./refex/ex60.pp

```
Program Example60;

{ Program to demonstrate the Seg function. }
Var
  W : Word;

begin
  W:=Seg(W);   { W contains its own Segment}
end.
```

29.9.276 Setjmp

Synopsis: Save current execution point.

Declaration: `function Setjmp(var S: jmp_buf) : LongInt`

Visibility: default

Description: `SetJmp` fills `env` with the necessary data for a jump back to the point where it was called. It returns zero if called in this way. If the function returns nonzero, then it means that a call to LongJmp (1263) with `env` as an argument was made somewhere in the program.

Errors: None.

See also: LongJmp (1263)

Listing: ./refex/ex79.pp

```
program example79;

{ Program to demonstrate the setjmp, longjmp functions }

procedure dojmp(var env : jmp_buf; value : longint);

begin
  value:=2;
  Writeln ('Going to jump !');
  { This will return to the setjmp call,
    and return value instead of 0 }
  longjmp(env,value);
end;

var env : jmp_buf;

begin
  if setjmp(env)=0 then
    begin
    writeln ('Passed first time.');
```

```
    dojmp(env,2);
    end
  else
    writeln ('Passed second time.');
end.
```

29.9.277 SetLength

Synopsis: Set length of a string.

Declaration:
```
procedure SetLength(var S: AStringType;Len: Integer)
procedure SetLength(var A: DynArrayType;Len: Integer)
```

Visibility: default

Description: `SetLength` sets the length of the string `S` to `Len`. `S` can be an ansistring, a short string or a widestring. For `ShortStrings`, `Len` can maximally be 255. For `AnsiStrings` it can have any value. For `AnsiString` strings, `SetLength` {\em must} be used to set the length of the string.

In the case of a dynamical array `A`, setlength sets the number of elements. The elements are numbered from index 0, so the count runs from 0 to `Len-1`. If Zero is specified, the array is cleared.

Errors: None.

See also: Length (1260)

Listing: ./refex/ex85.pp

```
Program Example85;

{ Program to demonstrate the SetLength function. }

Var S : String;

begin
  FillChar(S[1],100,#32);
  Setlength(S,100);
  Writeln ('"',S,'"');
end.
```

29.9.278 SetMemoryManager

Synopsis: Set a memory manager

Declaration:
```
procedure SetMemoryManager(const MemMgr: TMemoryManager)
```

Visibility: default

Description: `SetMemoryManager` sets the current memory manager record to `MemMgr`.

For an example, see \progref.

Errors: None.

See also: GetMemoryManager (1245), IsMemoryManagerSet (1259)

29.9.279 SetMemoryMutexManager

Synopsis: Procedure to set the mutex manager.

Declaration:
```
procedure SetMemoryMutexManager(var MutexMgr: TMemoryMutexManager)
```

Visibility: default

Description: `SetMemoryMutexManager` sets the mutex manager used by the memory manager to `MutexMgr`. The current mutex manager is returned in `MutexMgr`

Errors: None.

See also: TMemoryMutexManager (1194), SetMemoryManager (1311)

29.9.280 SetString

Synopsis: Set length of a string and copy buffer.

Declaration:
```
procedure SetString(out S: AnsiString;Buf: PChar;Len: SizeInt)
procedure SetString(out S: Shortstring;Buf: PChar;Len: SizeInt)
procedure SetString(out S: WideString;Buf: PWideChar;Len: SizeInt)
procedure SetString(out S: WideString;Buf: PChar;Len: SizeInt)
```

Visibility: default

Description: `SetString` sets the length of the string `S` to `Len` and if `Buf` is non-nil, copies `Len` characters from `Buf` into `S`. `S` can be an ansistring, a short string or a widestring. For `ShortStrings`, `Len` can maximally be 255.

Errors: None.

See also: SetLength (1311)

29.9.281 SetTextBuf

Synopsis: Set size of text file internal buffer

Declaration:
```
procedure SetTextBuf(var f: Text;var Buf)
procedure SetTextBuf(var f: Text;var Buf;Size: SizeInt)
```

Visibility: default

Description: `SetTextBuf` assigns an I/O buffer to a text file. The new buffer is located at `Buf` and is `Size` bytes long. If `Size` is omitted, then `SizeOf(Buf)` is assumed. The standard buffer of any text file is 128 bytes long. For heavy I/O operations this may prove too slow. The `SetTextBuf` procedure allows to set a bigger buffer for the I/O of the application, thus reducing the number of system calls, and thus reducing the load on the system resources. The maximum size of the newly assigned buffer is 65355 bytes.

Remark:

- Never assign a new buffer to an opened file. A new buffer can be assigned immediately after a call to Rewrite (1303), Reset (1302) or `Append`, but not after the file was read from/written to. This may cause loss of data. If a new buffer must be assigned after read/write operations have been performed, the file should be flushed first. This will ensure that the current buffer is emptied.

- Take care that the assigned buffer is always valid. If a local variable is assigned as a buffer, then after the program exits the local program block, the buffer will no longer be valid, and stack problems may occur.

Errors: No checking on `Size` is done.

See also: Assign (1210), Reset (1302), Rewrite (1303), Append (1208)

Listing: ./refex/ex61.pp

```
Program Example61;

{ Program to demonstrate the SetTextBuf function. }

Var
  Fin,Fout : Text;
  Ch : Char;
  Bufin,Bufout : Array[1..10000] of byte;

begin
  Assign (Fin,paramstr(1));
  Reset (Fin);
  Assign (Fout,paramstr(2));
  Rewrite (Fout);
  { This is harmless before IO has begun }
  { Try this program again on a big file,
    after commenting out the following 2
    lines and recompiling it. }
  SetTextBuf (Fin,Bufin);
  SetTextBuf (Fout,Bufout);
  While not eof(Fin) do
    begin
    Read (Fin,ch);
    write (Fout,ch);
    end;
  Close (Fin);
  Close (Fout);
end.
```

29.9.282 SetTextLineEnding

Synopsis: Set the end-of-line character for the given text file.

Declaration: `procedure SetTextLineEnding(var f: Text;Ending: String)`

Visibility: default

Description: `SetTextLineEnding` sets the end-of-line character for the text file `F` to `Ending`. By default, this is the string indicated by DefaultTextLineBreakStyle (1163).

Errors: None.

See also: DefaultTextLineBreakStyle (1163), TTextLineBreakStyle (1197)

29.9.283 SetThreadManager

Synopsis: Set the thread manager, optionally return the current thread manager.

Declaration:
```
function SetThreadManager(const NewTM: TThreadManager;
                          var OldTM: TThreadManager) : Boolean
function SetThreadManager(const NewTM: TThreadManager) : Boolean
```

Visibility: default

Description: `SetThreadManager` sets the thread manager to `NewTM`. If `OldTM` is given, `SetThreadManager` uses it to return the previously used thread manager.

The function returns `True` if the threadmanager was set succesfully, `False` if an error occurred.

For more information about thread programming, see the programmer's guide.

Errors: If an error occurred cleaning up the previous manager, or an error occurred initializing the new manager, `False` is returned.

See also: GetThreadManager (1245), TThreadManager (1198)

29.9.284 SetVariantManager

Synopsis: Set the current variant manager.

Declaration:
```
procedure SetVariantManager(const VarMgr: tvariantmanager)
```

Visibility: default

Description: `SetVariantManager` sets the variant manager to `varmgr`.

See also: IsVariantManagerSet (1159), GetVariantManager (1246)

29.9.285 SetWideStringManager

Synopsis: Set the widestring manager

Declaration:
```
procedure SetWideStringManager(const New: TWideStringManager)
procedure SetWideStringManager(const New: TWideStringManager;
                               var Old: TWideStringManager)
```

Visibility: default

Description: `SetWideStringManager` sets the current widestring manager to `New`. Optionally, it returns the currently active widestring manager in `Old`.

WideStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a WideString manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit cwstrings).

It is possible to implement a custom widestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

Errors:

See also: TWideStringManager (1202)

29.9.286 ShortCompareText

Synopsis: Compare 2 shortstrings

Declaration:
```
function ShortCompareText(const S1: shortstring;const S2: shortstring)
                                 : SizeInt
```

Visibility: default

Description: `ShortCompareText` compares two shortstrings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparision of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: #rtl.sysutils.CompareText (1391)

29.9.287 sin

Synopsis: Calculate sine of angle

Declaration:
```
function sin(d: ValReal) : ValReal
```

Visibility: default

Description: `Sin` returns the sine of its argument `X`, where `X` is an angle in radians. If the absolute value of the argument is larger than 2ĝ3, then the result is undefined.

Errors: None.

See also: Cos (1224), Pi (1294), Exp (1235), Ln (1261)

Listing: ./refex/ex62.pp

```
Program Example62;

{ Program to demonstrate the Sin function. }

begin
  Writeln (Sin(Pi):0:1);   { Prints 0.0 }
  Writeln (Sin(Pi/2):0:1); { Prints 1.0 }
end.
```

29.9.288 SizeOf

Synopsis: Return size of a variable or type.

Declaration:
```
function SizeOf(X: TAnyType) : LongInt
```

Visibility: default

Description: `SizeOf` returns the size, in bytes, of any variable or type-identifier.

Remark: This isn't really a RTL function. Its result is calculated at compile-time, and hard-coded in the executable.

Errors: None.

See also: Addr (1206)

Listing: ./refex/ex63.pp

```
Program Example63;

{ Program to demonstrate the SizeOf function. }
Var
  I : Longint;
  S : String [10];

begin
  WriteIn (SizeOf(I));  { Prints 4  }
  WriteIn (SizeOf(S));  { Prints 11 }
end.
```

29.9.289 SizeofResource

Synopsis: Return the size of a particular resource

Declaration:
```
function SizeofResource(ModuleHandle: HMODULE;
                        ResHandle: TResourceHandle) : Integer
```

Visibility: default

Description: `SizeOfResource` returns the size of the resource identified by `ResHandle` in module identified by `ModuleHandle`. `ResHandle` should be obtained from a call to LoadResource (1262)

Errors: In case of an error, 0 is returned.

See also: FindResource (1240), FreeResource (1243), LoadResource (1262), LockResource (1263), UnlockResource (1329), FreeResource (1243)

29.9.290 Space

Synopsis: Return a string of spaces

Declaration:
```
function Space(b: Byte) : shortstring
```

Visibility: default

Description: `Space` returns a shortstring with length `B`, consisting of spaces.

See also: StringOfChar (1319)

29.9.291 Sptr

Synopsis: Return current stack pointer

Declaration:
```
function Sptr : Pointer
```

Visibility: default

Description: `Sptr` returns the current stack pointer.

Errors: None.

See also: SSeg (1318)

Listing: ./refex/ex64.pp

```
program Example64;

{ Program to demonstrate the sptr function. }

var p:ptruint;

begin
  p:=ofs(sptr); { P Contains now the current stack position. }
end.
```

29.9.292 sqr

Synopsis: Calculate the square of a value.

Declaration:
```
function sqr(l: LongInt) : LongInt
function sqr(l: Int64) : Int64
function sqr(l: QWord) : QWord
function sqr(d: ValReal) : ValReal
```

Visibility: default

Description: `Sqr` returns the square of its argument `X`.

Errors: None.

See also: Sqrt (1318), Ln (1261), Exp (1235)

Listing: ./refex/ex65.pp

```
Program Example65;

{ Program to demonstrate the Sqr function. }
Var i : Integer;

begin
  For i:=1 to 10 do
    writeln (Sqr(i):3);
end.
```

29.9.293 sqrt

Synopsis: Calculate the square root of a value

Declaration: `function sqrt(d: ValReal) : ValReal`

Visibility: default

Description: `Sqrt` returns the square root of its argument `X`, which must be positive.

Errors: If `X` is negative, then a run-time error is generated.

See also: Sqr (1317), Ln (1261), Exp (1235)

Listing: ./refex/ex66.pp

```
Program Example66;

{ Program to demonstrate the Sqrt function. }

begin
  Writeln (Sqrt(4):0:3); { Prints 2.000 }
  Writeln (Sqrt(2):0:3); { Prints 1.414 }
end.
```

29.9.294 Sseg

Synopsis: Return stack segment register value.

Declaration: `function Sseg : Word`

Visibility: default

Description: `SSeg` returns the Stack Segment. This function is only supported for compatibility reasons, as `Sptr` returns the correct contents of the stackpointer.

Errors: None.

See also: Sptr (1317)

Listing: ./refex/ex67.pp

```
Program Example67;

{ Program to demonstrate the SSeg function. }
Var W : Longint;

begin
  W:=SSeg;
end.
```

29.9.295 Str

Synopsis: Convert a numerical value to a string.

Declaration: `procedure Str(var X: TNumericType;var S: String)`

Visibility: default

Description: `Str` returns a string which represents the value of X. X can be any numerical type. The actual declaration of `Str` is not according to pascal syntax, and should be

```
procedure Str(var X: TNumericType[:NumPlaces[:Decimals]];var S: String)
```

Where the optional `NumPlaces` and `Decimals` specifiers control the formatting of the string: `NumPlaces` gives the total width of the string, and `Decimals` the number of decimals after the decimal separator char.

Errors: None.

See also: Val (1331)

Listing: ./refex/ex68.pp

```
Program Example68;

{ Program to demonstrate the Str function. }
Var S : String;

Function IntToStr (I : Longint) : String;

Var S : String;

begin
 Str (I,S);
 IntToStr:=S;
end;

begin
  S:='*'+IntToStr(-233)+'*';
  WriteIn (S);
end.
```

29.9.296 StringOfChar

Synopsis: Return a string consisting of 1 character repeated N times.

Declaration: `function StringOfChar(c: Char;l: SizeInt) : AnsiString`

Visibility: default

Description: `StringOfChar` creates a new String of length `l` and fills it with the character `c`.

It is equivalent to the following calls:

```
SetLength(StringOfChar,l);
FillChar(Pointer(StringOfChar)^,Length(StringOfChar),c);
```

Errors: None.

See also: SetLength (1311)

Listing: ./refex/ex97.pp

```
Program Example97;

{$H+}

{ Program to demonstrate the StringOfChar function. }

Var S : String;

begin
  S:=StringOfChar(' ',40)+'Aligned at column 41.';
  Writeln(s);
end.
```

29.9.297 StringToPPChar

Synopsis: Split string in list of null-terminated strings

Declaration:
```
function StringToPPChar(var S: AnsiString;ReserveEntries: Integer)
                              : PPChar
function StringToPPChar(S: PChar;ReserveEntries: Integer) : PPChar
```

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of pchars that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function is only available on certain platforms.

Errors: None.

See also: ArrayStringToPPchar (1209)

29.9.298 StringToWideChar

Synopsis: Convert a string to an array of widechars.

Declaration:
```
function StringToWideChar(const Src: AnsiString;Dest: PWideChar;
                              DestSize: SizeInt) : PWideChar
```

Visibility: default

Description: `StringToWideChar` converts an ansistring `Src` to a null-terminated array of WideChars. The destination for this array is pointed to by `Dest`, and contains room for at least `DestSize` widechars.

Errors: No validity checking is performed on `Dest`.

See also: WideCharToString (1333), WideCharToStrVar (1333), WideCharLenToStrVar (1333), WideCharLenToString (1333)

29.9.299 strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: PChar) : LongInt`

Visibility: default

Description: Returns the length of the null-terminated string `P`.

Errors: None.

29.9.300 strpas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function strpas(p: PChar) : shortstring`

Visibility: default

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.

29.9.301 Succ

Synopsis: Return next element of ordinal type.

Declaration: `function Succ(X: TOrdinal) : TOrdinal`

Visibility: default

Description: `Succ` returns the element that succeeds the element that was passed to it. If it is applied to the last value of the ordinal type, and the program was compiled with range checking on (\var{\{\$R+\}}), then a run-time error will be generated.

for an example, see Ord (1292).

Errors: Run-time error 201 is generated when the result is out of range.

See also: Ord (1292), Pred (1296), High (1249), Low (1263)

29.9.302 SuspendThread

Synopsis: Suspend a running thread.

Declaration: `function SuspendThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `SuspendThread` suspends a running thread. The thread is identified with it's handle or ID `threadHandle`.

The function returns zero if succesful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: ResumeThread (1303), KillThread (1260)

29.9.303 Swap

Synopsis: Swap high and low bytes/words of a variable

Declaration:
```
function swap(X: Word) : Word
function Swap(X: Integer) : Integer
function swap(X: LongInt) : LongInt
function Swap(X: Cardinal) : Cardinal
function Swap(X: QWord) : QWord
function swap(X: Int64) : Int64
```

Visibility: default

Description: `Swap` swaps the high and low order bytes of `X` if `X` is of type `Word` or `Integer`, or swaps the high and low order words of `X` if `X` is of type `Longint` or `Cardinal`. The return type is the type of `X`

Errors: None.

See also: Lo (1262), Hi (1248)

Listing: ./refex/ex69.pp

```
Program Example69;

{ Program to demonstrate the Swap function. }
Var W : Word;
    L : Longint;

begin
  W:=$1234;
  W:=Swap(W);
  if W<>$3412 then
    writeln ('Error when swapping word !');
  L:=$12345678;
  L:=Swap(L);
  if L<>$56781234 then
    writeln ('Error when swapping Longint !');
end.
```

29.9.304 SwapEndian

Synopsis: Swap endianness of the argument

Declaration:
```
function SwapEndian(const AValue: SmallInt) : SmallInt
function SwapEndian(const AValue: Word) : Word
function SwapEndian(const AValue: LongInt) : LongInt
function SwapEndian(const AValue: DWord) : DWord
function SwapEndian(const AValue: Int64) : Int64
function SwapEndian(const AValue: QWord) : QWord
```

Visibility: default

Description: `SwapEndian` will swap the endianness of the bytes in its argument.

Errors: None.

See also: hi (1248), lo (1262), swap (1322), BEToN (1213), NToBE (1266), NToLE (1267), LEToN (1261)

29.9.305 SysAllocMem

Synopsis: System memory manager: Allocate memory

Declaration:
```
function SysAllocMem(size: PtrInt) : Pointer
```

Visibility: default

Description: `SysFreemSize` is the system memory manager implementation for AllocMem (1207)

See also: AllocMem (1207)

29.9.306 SysAssert

Synopsis: Standard Assert failure implementation

Declaration:
```
procedure SysAssert(const Msg: ShortString;const FName: ShortString;
                    LineNo: LongInt;ErrorAddr: Pointer)
```

Visibility: default

Description: `SysAssert` is the standard implementation of the assertion failed code. It is the default value of the `AssertErrorProc` constant. It will print the assert message `Msg` together with the filename `FName` and linenumber `LineNo` to standard error output (`StdErr`) and will halt the program with exit code 227. The error address `ErrorAddr` is ignored.

See also: AssertErrorProc (1162)

29.9.307 SysBackTraceStr

Synopsis: Format an address suitable for inclusion in a backtrace

Declaration:
```
function SysBackTraceStr(Addr: Pointer) : ShortString
```

Visibility: default

Description: `SysBackTraceStr` will create a string representation of the address `Addr`, suitable for inclusion in a stack backtrace.

Errors: None.

29.9.308 SysFreemem

Synopsis: System memory manager free routine.

Declaration:
```
function SysFreemem(p: pointer) : PtrInt
```

Visibility: default

Description: `SysFreeem` is the system memory manager implementation for FreeMem (1242)

See also: FreeMem (1242)

29.9.309 SysFreememSize

Synopsis: System memory manager free routine.

Declaration: `function SysFreememSize(p: pointer;Size: PtrInt) : PtrInt`

Visibility: default

Description: `SysFreemSize` is the system memory manager implementation for FreeMem (1242)

See also: MemSize (1264)

29.9.310 SysGetFPCHeapStatus

Synopsis: Return the status of the FPC heapmanager

Declaration: `function SysGetFPCHeapStatus : TFPCHeapStatus`

Visibility: default

Description: `SysGetFPCHeapStatus` returns the status of the default FPC heapmanager. It is set as the default value of the corresponding GetFPCHeapStatus (1244) function.

Errors: None. The result of this function is bogus information if the current heapmanager is not the standard FPC heapmanager.

See also: GetFPCHeapStatus (1244)

29.9.311 SysGetHeapStatus

Synopsis: System implementation of GetHeapStatus (1244)

Declaration: `function SysGetHeapStatus : THeapStatus`

Visibility: default

Description: `SysGetHeapStatus` is the system implementation of the GetHeapStatus (1244) call.

See also: GetHeapStatus (1244)

29.9.312 SysGetmem

Synopsis: System memory manager memory allocator.

Declaration: `function SysGetmem(Size: PtrInt) : Pointer`

Visibility: default

Description: `SysGetmem` is the system memory manager implementation for GetMem (1244)

See also: GetMem (1244), GetMemory (1244)

29.9.313 SysInitExceptions

Synopsis: Initialize exceptions.

Declaration: `procedure SysInitExceptions`

Visibility: default

Description: `SysInitExceptions` initializes the exception system. This procedure should never be called directly, it is taken care of by the RTL.

29.9.314 SysInitFPU

Synopsis: Initialize the FPU

Declaration: `procedure SysInitFPU`

Visibility: default

Description: `SysInitFPU` initializes (resets) the floating point unit, if one is available. It is called for instance when a new thread is started.

See also: BeginThread (1212)

29.9.315 SysInitStdIO

Synopsis: Initialize standard input and output.

Declaration: `procedure SysInitStdIO`

Visibility: default

Description: `SysInitStdIO` initializes the standard input and output files: Output (1204), Input (1204) and StdErr (1205). This routine is called by the initialization code of the system unit, there should be no need to call it directly.

29.9.316 SysMemSize

Synopsis: System memory manager: free size.

Declaration: `function SysMemSize(p: pointer) : PtrInt`

Visibility: default

Description: `SysFreemSize` is the system memory manager implementation for MemSize (1264)

See also: MemSize (1264)

29.9.317 SysReAllocMem

Synopsis: System memory manager: Reallocate memory

Declaration: `function SysReAllocMem(var p: pointer;size: PtrInt) : Pointer`

Visibility: default

Description: `SysReallocMem` is a help routine for the system memory manager implementation for ReAllocMem (1301).

See also: ReAllocMem (1301)

29.9.318 SysResetFPU

Synopsis: Reset the floating point unit.

Declaration: `procedure SysResetFPU`

Visibility: default

Description: `SysResetFPU` resets the floating point unit. There should normally be no need to call this unit; the compiler itself takes care of this.

29.9.319 SysSetCtrlBreakHandler

Synopsis: System CTRL-C handler

Declaration:
```
function SysSetCtrlBreakHandler(Handler: TCtrlBreakHandler)
                                      : TCtrlBreakHandler
```

Visibility: default

Description: `SysSetCtrlBreakHandler` sets the CTRL-C handler to the `Handler` callback, and returns the previous value of the handler.

See also: TCtrlBreakHandler (1191)

29.9.320 SysTryResizeMem

Synopsis: System memory manager: attempt to resize memory.

Declaration:
```
function SysTryResizeMem(var p: pointer;size: PtrInt) : Boolean
```

Visibility: default

Description: `SysTryResizeMem` is a help routine for the system memory manager implementation for ReAllocMem (1301), SysReAllocMem (1325)

See also: SysReAllocMem (1325), ReAllocMem (1301)

29.9.321 ThreadGetPriority

Synopsis: Return the priority of a thread.

Declaration:
```
function ThreadGetPriority(threadHandle: TThreadID) : LongInt
```

Visibility: default

Description: `ThreadGetPriority` returns the priority of thread `TThreadID` to `Prio`. The returned priority is a value between -15 and 15.

Errors: None.

See also: ThreadSetPriority (1326)

29.9.322 ThreadSetPriority

Synopsis: Set the priority of a thread.

Declaration:
```
function ThreadSetPriority(threadHandle: TThreadID;Prio: LongInt)
                                    : Boolean
```

Visibility: default

Description: `ThreadSetPriority` sets the priority of thread `TThreadID` to `Prio`. Priority is a value between -15 and 15.

Errors: None.

See also: ThreadGetPriority (1326)

29.9.323 ThreadSwitch

Synopsis: Signal possibility of thread switch

Declaration: `procedure ThreadSwitch`

Visibility: default

Description: `ThreadSwitch` signals the operating system that the thread should be suspended and that another thread should be executed.

This call is a hint only, and may be ignored.

See also: SuspendThread (1321), ResumeThread (1303), KillThread (1260)

29.9.324 trunc

Synopsis: Truncate a floating point value.

Declaration: `function trunc(d: ValReal) : Int64`

Visibility: default

Description: `Trunc` returns the integer part of `X`, which is always smaller than (or equal to) `X` in absolute value.

Errors: None.

See also: Frac (1241), Int (1256), Round (1305)

Listing: ./refex/ex70.pp

```
Program Example70;

{ Program to demonstrate the Trunc function. }

begin
  Writeln (Trunc(123.456)); { Prints 123  }
  Writeln (Trunc(-123.456)); { Prints -123 }
  Writeln (Trunc(12.3456)); { Prints 12   }
  Writeln (Trunc(-12.3456)); { Prints -12  }
end.
```

29.9.325 Truncate

Synopsis: Truncate the file at position

Declaration: `procedure Truncate(var F: File)`

Visibility: default

Description: `Truncate` truncates the (opened) file `F` at the current file position.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use `IOResult` to check for errors.

See also: Append (1208), Filepos (1236), Seek (1307)

Listing: ./refex/ex71.pp

```
Program Example71;

{ Program to demonstrate the Truncate function. }

Var F : File of longint;
    I,L : Longint;

begin
  Assign (F,'test.tmp');
  Rewrite (F);
  For I:=1 to 10 Do
    Write (F,I);
  Writeln ('Filesize before Truncate : ',FileSize(F));
  Close (f);
  Reset (F);
  Repeat
    Read (F,I);
  Until i=5;
  Truncate (F);
  Writeln ('Filesize after Truncate  : ',Filesize(F));
  Close (f);
end.
```

29.9.326 UCS4StringToWideString

Synopsis:

Declaration: `function UCS4StringToWideString(const s: UCS4String) : WideString`

Visibility: default

Description:

Errors:

29.9.327 Unassigned

Synopsis: Unassigned variant.

Declaration: `function Unassigned : Variant`

Visibility: default

29.9.328 UnicodeToUtf8

Synopsis:

Declaration:
```
function UnicodeToUtf8(Dest: PChar;Source: PWideChar;MaxBytes: SizeInt)
                        : SizeInt
function UnicodeToUtf8(Dest: PChar;MaxDestBytes: SizeUInt;
                       Source: PWideChar;SourceChars: SizeUInt)
                        : SizeUInt
```

Visibility: default

Description:

Errors:

29.9.329 UniqueString

Synopsis: Make sure reference count of string is 1

Declaration:
```
procedure UniqueString(var S: AnsiString)
procedure UniqueString(var S: WideString)
```

Visibility: default

Description: `UniqueString` ensures that the ansistring `S` has reference count 1. It makes a copy of `S` if this is necesary, and returns the copy in `S`

Errors: None.

29.9.330 UnlockResource

Synopsis: Unlock a previously locked resource

Declaration:
```
function UnlockResource(ResData: HGLOBAL) : LongBool
```

Visibility: default

Description: `UnlockResource` unlocks a previously locked resource. Note that this function does not exist on windows, it's only needed on other platforms.

Errors: The function returns `False` if it failed.

See also: FindResource (1240), FreeResource (1243), SizeofResource (1316), LoadResource (1262), lockResource (1263), FreeResource (1243)

29.9.331 upCase

Synopsis: Convert a string to all uppercase.

Declaration:
```
function upCase(const s: shortstring) : shortstring
function upCase(c: Char) : Char
function upcase(const s: ansistring) : ansistring
function UpCase(const s: WideString) : WideString
```

Visibility: default

Description: `Upcase` returns the uppercase version of its argument `C`. If its argument is a string, then the complete string is converted to uppercase. The type of the returned value is the same as the type of the argument.

Errors: None.

See also: Lowercase (1264)

Listing: ./refex/ex72.pp

```
program Example72;

{ Program to demonstrate the upcase function. }

var c:char;

begin
  for c:='a' to 'z' do
```

```
    write(upcase(c));
  WriteLn;
  { This doesn't work in TP, but it does in Free Pascal }
  WriteLn(upcase('abcdefghijklmnopqrstuvwxyz'));
end.
```

29.9.332 UTF8Decode

Synopsis:

Declaration: `function UTF8Decode(const s: UTF8String) : WideString`

Visibility: default

Description:

Errors:

29.9.333 UTF8Encode

Synopsis:

Declaration: `function UTF8Encode(const s: WideString) : UTF8String`

Visibility: default

Description:

Errors:

29.9.334 Utf8ToAnsi

Synopsis:

Declaration: `function Utf8ToAnsi(const s: UTF8String) : ansistring`

Visibility: default

Description:

Errors:

29.9.335 Utf8ToUnicode

Synopsis:

Declaration:
```
function Utf8ToUnicode(Dest: PWideChar;Source: PChar;MaxChars: SizeInt)
                            : SizeInt
function Utf8ToUnicode(Dest: PWideChar;MaxDestChars: SizeUInt;
                       Source: PChar;SourceBytes: SizeUInt) : SizeUInt
```

Visibility: default

Description:

Errors:

29.9.336 Val

Synopsis: Calculate numerical value of a string.

Declaration:
```
procedure Val(const S: String;var V;var Code: Word)
```

Visibility: default

Description: `Val` converts the value represented in the string `S` to a numerical value, and stores this value in the variable `V`, which can be of type `Longint`, `Real` and `Byte`. If the conversion isn't succesfull, then the parameter `Code` contains the index of the character in `S` which prevented the conversion. The string `S` is allowed to contain spaces in the beginning. The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference.

Errors: If the conversion doesn't succeed, the value of `Code` indicates the position where the conversion went wrong. The value of `V` is then undefined.

See also: Str (1318)

Listing: ./refex/ex74.pp

```
Program Example74;

{ Program to demonstrate the Val function. }
Var I, Code : Integer;

begin
  Val (ParamStr (1),I,Code);
  If Code<>0 then
    WriteIn ('Error at position ',code,' : ',Paramstr(1)[Code])
  else
    WriteIn ('Value : ',I);
end.
```

29.9.337 VarArrayGet

Synopsis:

Declaration:
```
function VarArrayGet(const A: Variant;const Indices: Array of SizeInt)
                          : Variant
```

Visibility: default

Description:

Errors:

29.9.338 VarArrayPut

Synopsis: Put a value in a single cell of a variant array

Declaration:
```
procedure VarArrayPut(var A: Variant;const Value: Variant;
                        const Indices: Array of SizeInt)
```

Visibility: default

Description: `VarArrayPut` puts `Value` in the variant array `A` at the location indicated by `Indices`. Thus the statement

```
VarArrayPut(A,B,[2,1]);
```

is equivalent to

```
A[2,1]:=B;
```

The difference is that the previous is usable when the amount of indices is not known at compile time.

Errors: If the number of indices is wrong (or out of range) an exception may be raised.

See also: VarArrayGet (1331)

29.9.339 VarArrayRedim

Synopsis: Redimension a variant array

Declaration:
```
procedure VarArrayRedim(var A: Variant;HighBound: SizeInt)
```

Visibility: default

Description: `VarArrayRedim` re-sizes the first dimension of the variant array `A`, giving it a new high bound `HighBound`. Obviously, `A` must be a variant array for this function to work.

Errors:

29.9.340 VarCast

Synopsis:

Declaration:
```
procedure VarCast(var dest: variant;const source: variant;
                  vartype: LongInt)
```

Visibility: default

Description:

Errors:

29.9.341 WaitForThreadTerminate

Synopsis: Wait for a thread to terminate.

Declaration:
```
function WaitForThreadTerminate(threadHandle: TThreadID;
                                TimeoutMs: LongInt) : DWord
```

Visibility: default

Description: `WaitForThreadTerminate` waits for a thread to finish its execution. The thread is identified by it's handle or ID `threadHandle`. If the thread does not exit within `TimeoutMs` milliseconds, the function will return with an error value.

The function returns the exit code of the thread.

See also: EndThread (1230), KillThread (1260)

29.9.342 WideCharLenToString

Synopsis: Convert a length-limited array of widechar to an ansistring

Declaration:
```
function WideCharLenToString(S: PWideChar;Len: SizeInt) : AnsiString
```

Visibility: default

Description: `WideCharLenToString` converts at most `Len` widecharacters from the null-terminated widechar array `S` to an ansistring, and returns the ansistring.

Errors: No validity checking is performed on `S`. Passing an invalid pointer may lead to access violations.

See also: StringToWideChar (1320), WideCharToString (1333), WideCharToStrVar (1333), WideCharLenToStrVar (1333)

29.9.343 WideCharLenToStrVar

Synopsis: Convert a length-limited array of widechar to an ansistring

Declaration:
```
procedure WideCharLenToStrVar(Src: PWideChar;Len: SizeInt;
                              out Dest: AnsiString)
```

Visibility: default

Description: `WideCharLenToString` converts at most `Len` widecharacters from the null-terminated widechar array `Src` to an ansistring, and returns the ansistring in `Dest`.

Errors: No validity checking is performed on `Src`. Passing an invalid pointer may lead to access violations.

See also: StringToWideChar (1320), WideCharToString (1333), WideCharToStrVar (1333), WideCharLenToString (1333)

29.9.344 WideCharToString

Synopsis: Convert a null-terminated array of widechar to an ansistring

Declaration:
```
function WideCharToString(S: PWideChar) : AnsiString
```

Visibility: default

Description: `WideCharToString` converts the null-terminated widechar array `S` to an ansistring, and returns the ansistring.

Errors: No validity checking is performed on `Src`. Passing an invalid pointer, or an improperly terminated array may lead to access violations.

See also: StringToWideChar (1320), WideCharToStrVar (1333), WideCharLenToStrVar (1333), WideCharLenToString (1333)

29.9.345 WideCharToStrVar

Synopsis: Convert a null-terminated array of widechar to an ansistring

Declaration:
```
procedure WideCharToStrVar(S: PWideChar;out Dest: AnsiString)
```

Visibility: default

Description: `WideCharToString` converts the null-terminated widechar array `S` to an ansistring, and returns the ansistring in `Dest`.

Errors: No validity checking is performed on `S`. Passing an invalid pointer, or an improperly terminated array may lead to access violations.

See also: StringToWideChar (1320), WideCharToString (1333), WideCharToStrVar (1333), WideCharLenToString (1333)

29.9.346 WideStringToUCS4String

Synopsis: Convert a widestring to a UCS-4 encoded string.

Declaration:
```
function WideStringToUCS4String(const s: WideString) : UCS4String
```

Visibility: default

Description: Convert a widestring to a UCS-4 encoded string.

Errors:

29.9.347 Write

Synopsis: Write variable to a text file

Declaration:
```
procedure Write(Args: Arguments)
procedure Write(var F: Text;Args: Arguments)
```

Visibility: default

Description: `Write` writes the contents of the variables `V1`, `V2` etc. to the file `F`. `F` can be a typed file, or a `Text` file. If `F` is a typed file, then the variables `V1`, `V2` etc. must be of the same type as the type in the declaration of `F`. Untyped files are not allowed. If the parameter `F` is omitted, standard output is assumed. If `F` is of type `Text`, then the necessary conversions are done such that the output of the variables is in human-readable format. This conversion is done for all numerical types. Strings are printed exactly as they are in memory, as well as `PChar` types. The format of the numerical conversions can be influenced through the following modifiers: `OutputVariable : NumChars [: Decimals ]` This will print the value of `OutputVariable` with a minimum of `NumChars` characters, from which `Decimals` are reserved for the decimals. If the number cannot be represented with `NumChars` characters, `NumChars` will be increased, until the representation fits. If the representation requires less than `NumChars` characters then the output is filled up with spaces, to the left of the generated string, thus resulting in a right-aligned representation. If no formatting is specified, then the number is written using its natural length, with nothing in front of it if it's positive, and a minus sign if it's negative. Real numbers are, by default, written in scientific notation.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the \var{\{\$i\}} switch.

See also: WriteLn (1335), Read (1298), Readln (1299), Blockwrite (1214)

29.9.348 WriteBarrier

Synopsis: Memory write barrier

Declaration:
```
procedure WriteBarrier
```

Visibility: default

Description: `WriteBarrier` is a low-level instruction to force a write barrier in the CPU: write (store) operations before and after the barrier are separate.

See also: ReadBarrier (1299), ReadDependencyBarrier (1299), ReadWriteBarrier (1300)

29.9.349 WriteLn

Synopsis: Write variable to a text file and append newline

Declaration:
```
procedure Writeln(Args: Arguments)
procedure WriteLn(var F: Text;Args: Arguments)
```

Visibility: default

Description: `WriteLn` does the same as Write (1334) for text files, and emits a Carriage Return - LineFeed character pair after that. If the parameter `F` is omitted, standard output is assumed. If no variables are specified, a Carriage Return - LineFeed character pair is emitted, resulting in a new line in the file `F`.

Remark: Under linux and unix, the Carriage Return character is omitted, as customary in Unix environments.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the \var{\{\$i\}} switch.

See also: Write (1334), Read (1298), Readln (1299), Blockwrite (1214)

Listing: ./refex/ex75.pp

```
Program Example75;

{ Program to demonstrate the Write(ln) function. }

Var
  F : File of Longint;
  L : Longint;

begin
  Write ('This is on the first line ! '); { No CR/LF pair! }
  Writeln ('And this too...');
  Writeln ('But this is already on the second line...');
  Assign (f,'test.tmp');
  Rewrite (f);
  For L:=1 to 10 do
    write (F,L); { No writeln allowed here ! }
  Close (f);
end.
```

29.10 IDispatch

29.10.1 Description

`IDispatch` is the pascal definition of the Windows Dispatch interface definition.

29.10.2 Method overview

Page	Property	Description
1336	GetIDsOfNames	Return IDs of named procedures
1336	GetTypeInfo	Return type information about properties
1336	GetTypeInfoCount	Return number of properties.
1336	Invoke	Invoke a dispatch method

29.10.3 IDispatch.GetTypeInfoCount

Synopsis: Return number of properties.

Declaration:
```
function GetTypeInfoCount(out count: LongInt) : HRESULT
```

Visibility: default

29.10.4 IDispatch.GetTypeInfo

Synopsis: Return type information about properties

Declaration:
```
function GetTypeInfo(Index: LongInt;LocaleID: LongInt;out TypeInfo)
                          : HRESULT
```

Visibility: default

29.10.5 IDispatch.GetIDsOfNames

Synopsis: Return IDs of named procedures

Declaration:
```
function GetIDsOfNames(const iid: TGuid;names: Pointer;
                            NameCount: LongInt;LocaleID: LongInt;
                            DispIDs: Pointer) : HRESULT
```

Visibility: default

Description: Return the ID of a procedure.

29.10.6 IDispatch.Invoke

Synopsis: Invoke a dispatch method

Declaration:
```
function Invoke(DispID: LongInt;const iid: TGuid;LocaleID: LongInt;
                     Flags: Word;var params;VarResult: pointer;
                     ExcepInfo: pointer;ArgErr: pointer) : HRESULT
```

Visibility: default

29.11 IInvokable

29.11.1 Description

`IInvokable` is a descendent of IInterface (1183), compiled in the `{$M+}` state, so Run-Time Type Information (RTTI) is generated for it.

29.12 IUnknown

29.12.1 Description

`IUnknown` is defined by windows. It's the basic interface which all COM objects must implement. The definition does not contain any code.

29.12.2 Method overview

Page	Property	Description
1337	_AddRef	Increase reference count of the interface
1337	_Release	Decrease reference count of the interface
1337	QueryInterface	Return pointer to VMT table of interface

29.12.3 IUnknown.QueryInterface

Synopsis: Return pointer to VMT table of interface

Declaration: `function QueryInterface(const iid: TGuid;out obj) : LongInt`

Visibility: default

29.12.4 IUnknown._AddRef

Synopsis: Increase reference count of the interface

Declaration: `function _AddRef : LongInt`

Visibility: default

See also: IUnknown._Release (1337)

29.12.5 IUnknown._Release

Synopsis: Decrease reference count of the interface

Declaration: `function _Release : LongInt`

Visibility: default

See also: IUnknown._AddRef (1337)

29.13 TAggregatedObject

29.13.1 Description

`TAggregatedObject` implements an object whose lifetime is governed by an external object (or interface). It does not implement the `IUnknown` interface by itself, but delegates all methods to the controller object, as exposed in the Controller (1338) property. In effect, the reference count of the aggregated object is the same as that of it's controller, and additionally, all interfaces of the controller are exposed by the aggregated object.

Note that the aggregated object maintains a non-counted reference to the controller.

Aggregated objects should be used when using delegation to implement reference counted objects: the delegated interfaces can be implemented safely by `TAggregatedObject` descendents.

29.13.2 Method overview

Page	Property	Description
1338	Create	Create a new instance of `TAggregatedObject`

29.13.3 Property overview

Page	Property	Access	Description
1338	Controller	r	Controlling instance

29.13.4 TAggregatedObject.Create

Synopsis: Create a new instance of `TAggregatedObject`

Declaration: `constructor Create(const aController: IUnknown)`

Visibility: public

Description: `Create` creates a new instance of `TAggregatedObject` on the heap, and stores a reference to `aController`, so it can be exposed in the Controller (1338) property.

Errors: If not enough memory is present on the heap, an exception will be raised. If the `aController` is `Nil`, exceptions will occur when any of the `TAggregatedObject` methods (actually, the `IUnknown` methods) are used.

See also: TAggregatedObject.Controller (1338)

29.13.5 TAggregatedObject.Controller

Synopsis: Controlling instance

Declaration: `Property Controller : IUnknown`

Visibility: public

Access: Read

Description: `Controller` exposes the controlling object, with all interfaces it has.

The value of the controller is set when the `TAggregatedObject` instance is created.

See also: TAggregatedObject.Create (1338)

29.14 TContainedObject

29.14.1 Description

`TContainedObject` is the base class for contained objects, i.e. objects that do not implement a reference counting mechanism themselves, but are owned by some other object which handles the reference counting mechanism. It implements the `IUnknown` interface and, more specifically, the `QueryInterface` method of `IUnknown`.

29.15 TInterfacedObject

29.15.1 Description

`TInterfacedObject` is a descendent of TObject (1340) which implements the IUnknown (1337) interface. It can be used as a base class for all classes which need reference counting.

29.15.2 Method overview

Page	Property	Description
1339	AfterConstruction	Handle reference count properly.
1339	BeforeDestruction	Check reference count.
1339	NewInstance	Create a new instance

29.15.3 Property overview

Page	Property	Access	Description
1340	RefCount	r	Return the current reference count

29.15.4 TInterfacedObject.AfterConstruction

Synopsis: Handle reference count properly.

Declaration: `procedure AfterConstruction;  Override`

Visibility: public

Description: `AfterConstruction` overrides the basic method in `TObject` and adds some additional reference count handling.

Errors: None.

See also: TInterfacedObject.BeforeDestruction (1339)

29.15.5 TInterfacedObject.BeforeDestruction

Synopsis: Check reference count.

Declaration: `procedure BeforeDestruction;  Override`

Visibility: public

Description: `AfterConstruction` overrides the basic method in `TObject` and adds a reference count check: if the reference count is not zero, an error occurs.

Errors: A runtime-error 204 will be generated if the reference count is nonzero when the object is destroyed.

See also: TInterfacedObject.AfterConstruction (1339)

29.15.6 TInterfacedObject.NewInstance

Synopsis: Create a new instance

Declaration: `function NewInstance : TObject;  Override`

Visibility: public

Description: `NewInstance` initializes a new instance of TInterfacedObject (1339)

Errors: None.

29.15.7 TInterfacedObject.RefCount

Synopsis: Return the current reference count

Declaration: `Property RefCount : LongInt`

Visibility: public

Access: Read

Description: `RefCount` returns the current reference count. This reference count cannot be manipulated, except through the methods of IUnknown (1337). When it reaches zero, the class instance is destroyed.

See also: IUnknown (1337)

29.16 TObject

29.16.1 Description

`TObject` is the parent root class for all classes in Object Pascal. If a class has no parent class explicitly declared, it is dependent on `TObject`. `TObject` introduces class methods that deal with the class' type information, and contains all necessary methods to create an instance at runtime, and to dispatch messages to the correct method (both string and integer messages).

29.16.2 Method overview

Page	Property	Description
1347	AfterConstruction	Method called after the constructor was called.
1347	BeforeDestruction	Method called before the destructor is called.
1344	ClassInfo	Return a pointer to the type information for this class.
1344	ClassName	Return the current class name.
1344	ClassNameIs	Check whether the class name equals the given name.
1345	ClassParent	Return the parent class.
1344	ClassType	Return a "class of" pointer for the current class
1343	CleanupInstance	Finalize the class instance.
1341	Create	`TObject` Constructor
1343	DefaultHandler	Default handler for integer message handlers.
1347	DefaultHandlerStr	Default handler for string messages.
1341	Destroy	`TObject` destructor.
1346	Dispatch	Dispatch an integer message
1346	DispatchStr	Dispatch a string message.
1347	FieldAddress	Return the address of a field.
1343	Free	Check for `Nil` and call destructor.
1342	FreeInstance	Clean up instance and free the memory reserved for the instance.
1348	GetInterface	Return a reference to an interface
1348	GetInterfaceByStr	
1348	GetInterfaceEntry	
1348	GetInterfaceEntryByStr	
1349	GetInterfaceTable	
1345	InheritsFrom	Chck wether class is an ancestor.
1343	InitInstance	Initialize a new class instance.
1345	InstanceSize	Return the size of an instance.
1346	MethodAddress	Return the address of a method
1346	MethodName	Return the name of a method.
1342	newinstance	Allocate memory on the heap for a new instance
1342	SafeCallException	Handle exception object
1345	StringMessageTable	Return a pointer to the string message table.

29.16.3 TObject.Create

Synopsis: `TObject` Constructor

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of `TObject`. Currently it does nothing. It is also not virtual, so there is in principle no need to call it directly.

See also: TObject.Destroy (1341)

29.16.4 TObject.Destroy

Synopsis: `TObject` destructor.

Declaration: `destructor Destroy;  Virtual`

Visibility: public

Description: `Destroy` is the destructor of `TObject`. It will clean up the memory assigned to the instance. Descendent classes should override destroy if they want to do additional clean-up. No other destructor should be implemented.

It is bad programming practice to call `Destroy` directly. It is better to call the Free (1343) method, because that one will check first if `Self` is different from `Nil`.

To clean up an instance and reset the refence to the instance, it is best to use the FreeAndNil (1435) function.

See also: TObject.Create (1341), TObject.Free (1343)

29.16.5 TObject.newinstance

Synopsis: Allocate memory on the heap for a new instance

Declaration:
```
function newinstance : TObject;  Virtual
```

Visibility: public

Description: `NewInstance` allocates memory on the heap for a new instance of the current class. If the memory was allocated, the class will be initialized by a call to InitInstance (1343). The function returns the newly initialized instance.

Errors: If not enough memory is available, a `Nil` pointer may be returned, or an exception may be raised.

See also: TObject.Create (1341), TObject.InitInstance (1343), TObject.InstanceSize (1345), TObject.FreeInstance (1342)

29.16.6 TObject.FreeInstance

Synopsis: Clean up instance and free the memory reserved for the instance.

Declaration:
```
procedure FreeInstance;  Virtual
```

Visibility: public

Description: `FreeInstance` cleans up an instance of the current class, and releases the heap memory occupied by the class instance.

See also: TObject.Destroy (1341), TObject.InitInstance (1343), TObject.NewInstance (1342)

29.16.7 TObject.SafeCallException

Synopsis: Handle exception object

Declaration:
```
function SafeCallException(exceptobject: TObject;exceptaddr: pointer)
                                  : LongInt;  Virtual
```

Visibility: public

Description: `SafeCallException` should be overridden to handle exceptions in a method marked with the savecall directive. The implementation in `TObject` simply returns zero.

29.16.8 TObject.DefaultHandler

Synopsis: Default handler for integer message handlers.

Declaration: `procedure DefaultHandler(var message); Virtual`

Visibility: public

Description: `DefaultHandler` is the default handler for messages. If a message has an unknown message ID (i.e. does not appear in the table with integer message handlers), then it will be passed to `DefaultHandler` by the Dispatch (1346) method.

Errors:

See also: TObject.Dispatch (1346), TObject.DefaultHandlerStr (1347)

29.16.9 TObject.Free

Synopsis: Check for `Nil` and call destructor.

Declaration: `procedure Free`

Visibility: public

Description: `Free` will check the `Self` pointer and calls Destroy (1341) if it is different from `Nil`. This is a safer method than calling `Destroy` directly. If a reference to the object must be reset as well (a recommended technque), then the function FreeAndNil (1435) should be called.

Errors: None.

See also: TObject.Destroy (1341), #rtl.sysutils.freeandnil (1435)

29.16.10 TObject.InitInstance

Synopsis: Initialize a new class instance.

Declaration: `function InitInstance(instance: pointer) : TObject`

Visibility: public

Description: `InitInstance` initializes the memory pointer to by `Instance`. This means that the VMT is initialized, and the interface pointers are set up correctly. The function returns the newly initialized instance.

See also: TObject.NewInstance (1342), TObject.Create (1341)

29.16.11 TObject.CleanupInstance

Synopsis: Finalize the class instance.

Declaration: `procedure CleanupInstance`

Visibility: public

Description: `CleanUpinstance` finalizes the instance, i.e. takes care of all reference counted objects, by decreasing their reference count by 1, and freeing them if their count reaches zero.

Normally, `CleanupInstance` should never be called, it is called automatically when the object is freed with it's constructor.

Errors: None.

See also: TObject.Destroy (1341), TObject.Free (1343), TObject.InitInstance (1343)

29.16.12 TObject.ClassType

Synopsis: Return a "class of" pointer for the current class

Declaration: `function ClassType : TClass`

Visibility: public

Description: `ClassType` returns a TClass (1191) class type reference for the current class.

See also: TClass (1191), TObject.ClassInfo (1344), TObject.ClassName (1344)

29.16.13 TObject.ClassInfo

Synopsis: Return a pointer to the type information for this class.

Declaration: `function ClassInfo : pointer`

Visibility: public

Description: `ClassInfo` returns a pointer to the type information for this class. This pointer can be used in the various type information routines.

29.16.14 TObject.ClassName

Synopsis: Return the current class name.

Declaration: `function ClassName : shortstring`

Visibility: public

Description: `ClassName` returns the class name for the current class, in all-uppercase letters. To check for the class name, use the ClassNameIs (1344) class method.

Errors: None.

See also: TObject.ClassInfo (1344), TObject.ClassType (1344), TObject.ClassNameIs (1344)

29.16.15 TObject.ClassNameIs

Synopsis: Check whether the class name equals the given name.

Declaration: `function ClassNameIs(const name: String) : Boolean`

Visibility: public

Description: `ClassNameIs` checks whether `Name` equals the class name. It takes of case sensitivity, i.e. it converts both names to uppercase before comparing.

See also: TObject.ClassInfo (1344), TObject.ClassType (1344), TObject.ClassName (1344)

29.16.16 TObject.ClassParent

Synopsis: Return the parent class.

Declaration: `function ClassParent : TClass`

Visibility: public

Description: `ClassParent` returns the class of the parent class of the current class. This is always different from `Nil`, except for `TObject`.

Errors: None.

See also: TObject.ClassInfo (1344), TObject.ClassType (1344), TObject.ClassName (1344)

29.16.17 TObject.InstanceSize

Synopsis: Return the size of an instance.

Declaration: `function InstanceSize : SizeInt`

Visibility: public

Description: `InstanceSize` returns the number of bytes an instance takes in memory. This is Just the memory occupied by the class structure, and does not take into account any additional memory that might be allocated by the constructor of the class.

Errors: None.

See also: TObject.InitInstance (1343), TObject.ClassName (1344), TObject.ClassInfo (1344), TObject.ClassType (1344)

29.16.18 TObject.InheritsFrom

Synopsis: Chck wether class is an ancestor.

Declaration: `function InheritsFrom(aclass: TClass) : Boolean`

Visibility: public

Description: `InheritsFrom` returns `True` if `AClass` is an ancestor class from the current class, and returns false if it is not.

Errors:

See also: TObject.ClassName (1344), TObject.ClassInfo (1344), TObject.ClassType (1344), TClass (1191)

29.16.19 TObject.StringMessageTable

Synopsis: Return a pointer to the string message table.

Declaration: `function StringMessageTable : pstringmessagetable`

Visibility: public

Description: `StringMessageTable` returns a pointer to the string message table, which can be used to look up methods for dispatching a string message. It is used by the DispatchStr (1346) method.

Errors: If there are no string message handlers, nil is returned.

See also: TObject.DispatchStr (1346), TObject.Dispatch (1346)

29.16.20 TObject.Dispatch

Synopsis: Dispatch an integer message

Declaration: `procedure Dispatch(var message)`

Visibility: public

Description: `Dispatch` looks in the message handler table for a handler that handles `message`. The message is identified by the first dword (cardinal) in the message structure.

If no matching message handler is found, the message is passed to the DefaultHandler (1343) method, which can be overridden by descendent classes to add custom handling of messages.

See also: TObject.DispatchStr (1346), TObject.DefaultHandler (1343)

29.16.21 TObject.DispatchStr

Synopsis: Dispatch a string message.

Declaration: `procedure DispatchStr(var message)`

Visibility: public

Description: `DispatchStr` extracts the message identifier from `Message` and checks the message handler table to see if a handler for the message is found, and calls the handler, passing along the message. If no handler is found, the default DefaultHandlerStr (1347) is called.

Errors: None.

See also: TObject.DefaultHandlerStr (1347), TObject.Dispatch (1346), TObject.DefaultHandler (1343)

29.16.22 TObject.MethodAddress

Synopsis: Return the address of a method

Declaration: `function MethodAddress(const name: shortstring) : pointer`

Visibility: public

Description: `MethodAddress` returns the address of a method, searching the method by its name. The `Name` parameter specifies which method should be taken. The search is conducted in a case-insensitive manner.

Errors: If no matching method is found, `Nil` is returned.

See also: TObject.MethodName (1346), TObject.FieldAddress (1347)

29.16.23 TObject.MethodName

Synopsis: Return the name of a method.

Declaration: `function MethodName(address: pointer) : shortstring`

Visibility: public

Description: `MethodName` searches the VMT for a method with the specified `address` and returns the name of the method.

Errors: If no method with the matching address is found, an empty string is returned.

See also: TObject.MethodAddress (1346), TObject.FieldAddress (1347)

29.16.24 TObject.FieldAddress

Synopsis: Return the address of a field.

Declaration: `function FieldAddress(const name: shortstring) : pointer`

Visibility: public

Description: `FieldAddress` returns the address of the field with name `name`. The address is the address of the field in the current class instance.

Errors: If no field with the specified name is found, `Nil` is returned.

See also: TObject.MethodAddress (1346), TObject.MethodName (1346)

29.16.25 TObject.AfterConstruction

Synopsis: Method called after the constructor was called.

Declaration: `procedure AfterConstruction;  Virtual`

Visibility: public

Description: `AfterConstruction` is a method called after the constructor was called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed after the constructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: TObject.BeforeDestruction (1347), TObject.Create (1341)

29.16.26 TObject.BeforeDestruction

Synopsis: Method called before the destructor is called.

Declaration: `procedure BeforeDestruction;  Virtual`

Visibility: public

Description: `BeforeDestruction` is a method called before the destructor is called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed before the destructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: TObject.AfterConstruction (1347), TObject.Destroy (1341), TObject.Free (1343)

29.16.27 TObject.DefaultHandlerStr

Synopsis: Default handler for string messages.

Declaration: `procedure DefaultHandlerStr(var message);  Virtual`

Visibility: public

Description: `DefaultHandlerStr` is called for string messages which have no handler associated with them in the string message handler table. The implementation of `DefaultHandlerStr` in `TObject` does nothing and mut be overridden by descendent classes to provide specific message handling behaviour.

See also: TObject.DispatchStr (1346), TObject.Dispatch (1346), TObject.DefaultHandler (1343)

29.16.28 TObject.GetInterface

Synopsis: Return a reference to an interface

Declaration: `function GetInterface(const iid: TGuid;out obj) : Boolean`

Visibility: public

Description: `GetInterface` scans the interface tables and returns a reference to the interface `iid`. The reference is stored in `Obj` which should be an interface reference. It returns `True` if the interface was found, `False` if not.

The reference count of the interface is increased by this call.

Errors: If no interface was found, `False` is returned.

See also: TObject.GetInterfaceByStr (1348)

29.16.29 TObject.GetInterfaceByStr

Synopsis:

Declaration: `function GetInterfaceByStr(const iidstr: String;out obj) : Boolean`

Visibility: public

Description:

Errors:

29.16.30 TObject.GetInterfaceEntry

Synopsis:

Declaration: `function GetInterfaceEntry(const iid: TGuid) : pinterfaceentry`

Visibility: public

Description:

Errors:

29.16.31 TObject.GetInterfaceEntryByStr

Synopsis:

Declaration: `function GetInterfaceEntryByStr(const iidstr: String) : pinterfaceentry`

Visibility: public

Description:

Errors:

29.16.32 TObject.GetInterfaceTable

Synopsis:

Declaration: `function GetInterfaceTable : pinterfacetable`

Visibility: public

Description:

Errors:

Chapter 30

Reference for unit 'sysutils'

30.1 Miscellaneous conversion routines

Functions for various conversions.

Table 30.1:

Name	Description
BCDToInt (1387)	Convert BCD number to integer
CompareMem (1389)	Compare two memory regions
FloatToStrF (1423)	Convert float to formatted string
FloatToStr (1422)	Convert float to string
FloatToText (1425)	Convert float to string
FormatFloat (1434)	Format a floating point value
GetDirs (1437)	Split string in list of directories
IntToHex (1443)	return hexadecimal representation of integer
IntToStr (1444)	return decumal representation of integer
StrToIntDef (1475)	Convert string to integer with default value
StrToInt (1474)	Convert string to integer
StrToFloat (1472)	Convert string to float
TextToFloat (1478)	Convert null-terminated string to float

30.2 Date/time routines

Functions for date and time handling.

30.3 FileName handling routines

Functions for file manipulation.

30.4 File input/output routines

Functions for reading/writing to file.

Table 30.2:

Name	Description
DateTimeToFileDate (1394)	Convert DateTime type to file date
DateTimeToStr (1395)	Construct string representation of DateTime
DateTimeToString (1395)	Construct string representation of DateTime
DateTimeToSystemTime (1396)	Convert DateTime to system time
DateTimeToTimeStamp (1397)	Convert DateTime to timestamp
DateToStr (1397)	Construct string representation of date
Date (1394)	Get current date
DayOfWeek (1398)	Get day of week
DecodeDate (1398)	Decode DateTime to year month and day
DecodeTime (1399)	Decode DateTime to hours, minutes and seconds
EncodeDate (1403)	Encode year, day and month to DateTime
EncodeTime (1404)	Encode hours, minutes and seconds to DateTime
FormatDateTime (1433)	Return string representation of DateTime
IncMonth (1442)	Add 1 to month
IsLeapYear (1445)	Determine if year is leap year
MSecsToTimeStamp (1448)	Convert nr of milliseconds to timestamp
Now (1449)	Get current date and time
StrToDateTime (1471)	Convert string to DateTime
StrToDate (1470)	Convert string to date
StrToTime (1476)	Convert string to time
SystemTimeToDateTime (1478)	Convert system time to datetime
TimeStampToDateTime (1480)	Convert time stamp to DateTime
TimeStampToMSecs (1481)	Convert Timestamp to number of millicseconds
TimeToStr (1481)	return string representation of Time
Time (1479)	Get current tyme

30.5 PChar related functions

Most PChar functions are the same as their counterparts in the STRINGS unit. The following functions are the same :

1. StrCat (1456) : Concatenates two `PChar` strings.
2. StrComp (1457) : Compares two `PChar` strings.
3. StrCopy (1458) : Copies a `PChar` string.
4. StrECopy (1459) : Copies a `PChar` string and returns a pointer to the terminating null byte.
5. StrEnd (1459) : Returns a pointer to the terminating null byte.
6. StrIComp (1460) : Case insensitive compare of 2 `PChar` strings.
7. StrLCat (1462) : Appends at most L characters from one `PChar` to another `PChar`.
8. StrLComp (1462) : Case sensitive compare of at most L characters of 2 `PChar` strings.
9. StrLCopy (1463) : Copies at most L characters from one `PChar` to another.
10. StrLen (1464) : Returns the length (exclusive terminating null byte) of a `PChar` string.
11. StrLIComp (1465) : Case insensitive compare of at most L characters of 2 `PChar` strings.

Table 30.3:

Name	Description
AddDisk (1371)	Add sisk to list of disk drives
ChangeFileExt (1389)	Change extension of file name
CreateDir (1392)	Create a directory
DeleteFile (1400)	Delete a file
DiskFree (1401)	Free space on disk
DiskSize (1401)	Total size of disk
ExpandFileName (1406)	Create full file name
ExpandUNCFileName (1407)	Create full UNC file name
ExtractFileDir (1407)	Extract drive and directory part of filename
ExtractFileDrive (1408)	Extract drive part of filename
ExtractFileExt (1408)	Extract extension part of filename
ExtractFileName (1409)	Extract name part of filename
ExtractFilePath (1409)	Extrct path part of filename
ExtractRelativePath (1409)	Construct relative path between two files
FileAge (1410)	Return file age
FileDateToDateTime (1412)	Convert file date to system date
FileExists (1413)	Determine whether a file exists on disk
FileGetAttr (1413)	Get attributes of file
FileGetDate (1414)	Get date of last file modification
FileSearch (1416)	Search for file in path
FileSetAttr (1417)	Get file attributes
FileSetDate (1418)	Get file dates
FindFirst (1419)	Start finding a file
FindNext (1420)	Find next file
GetCurrentDir (1437)	Return current working directory
RemoveDir (1451)	Remove a directory from disk
RenameFile (1451)	Rename a file on disk
SetCurrentDir (1453)	Set current working directory
SetDirSeparators (1454)	Set directory separator characters
FindClose (1419)	Stop searching a file
DoDirSeparators (1402)	Replace directory separator characters

12. StrLower (1465) : Converts a `PChar` to all lowercase letters.
13. StrMove (1466) : Moves one `PChar` to another.
14. StrNew (1466) : Makes a copy of a `PChar` on the heap, and returns a pointer to this copy.
15. StrPos (1468) : Returns the position of one `PChar` string in another?
16. StrRScan (1468) : returns a pointer to the last occurrence of on `PChar` string in another one.
17. StrScan (1469) : returns a pointer to the first occurrence of on `PChar` string in another one.
18. StrUpper (1477) : Converts a `PChar` to all uppercase letters.

The subsequent functions are different from their counterparts in STRINGS, although the same examples can be used.

Table 30.4:

Name	Description
FileCreate (1411)	Create a file and return handle
FileOpen (1415)	Open file end return handle
FileRead (1416)	Read from file
FileSeek (1417)	Set file position
FileTruncate (1418)	Truncate file length
FileWrite (1418)	Write to file
FileClose (1411)	Close file handle

30.6 Localization support

Localization support depends on various constants and structures being initialized correctly. On Windows and OS/2 this is done automatically: a widestring manager is installed by default which helps taking care of the current locale when performing various operations on strings. The various internationalization settings (date/time format, currency, language etc) are also initialized correctly on these platforms.

On unixes, the widestring support is in a separate unit: cwstrings, which loads the various needed functions from the C library. It should be added manually to the uses clause of your program. No initialization settings are applied by this unit, these must be initialized separately for the moment.

30.7 Formatting strings

Functions for formatting strings.

Table 30.5:

Name	Description
AdjustLineBreaks (1372)	Convert line breaks to line breaks for system
FormatBuf (1432)	Format a buffer
Format (1427)	Format arguments in string
FmtStr (1426)	Format buffer
QuotedStr (1450)	Quote a string
StrFmt (1460)	Format arguments in a string
StrLFmt (1464)	Format maximum L characters in a string
TrimLeft (1482)	Remove whitespace at the left of a string
TrimRight (1483)	Remove whitespace at the right of a string
Trim (1481)	Remove whitespace at both ends of a string

30.8 String functions

Functions for handling strings.

Table 30.6:

Name	Description
AnsiCompareStr (1373)	Compare two strings
AnsiCompareText (1374)	Compare two strings, case insensitive
AnsiExtractQuotedStr (1375)	Removes quotes from string
AnsiLastChar (1376)	Get last character of string
AnsiLowerCase (1376)	Convert string to all-lowercase
AnsiQuotedStr (1377)	Qoutes a string
AnsiStrComp (1378)	Compare strings case-sensitive
AnsiStrIComp (1379)	Compare strings case-insensitive
AnsiStrLComp (1381)	Compare L characters of strings case sensitive
AnsiStrLIComp (1381)	Compare L characters of strings case insensitive
AnsiStrLastChar (1380)	Get last character of string
AnsiStrLower (1382)	Convert string to all-lowercase
AnsiStrUpper (1384)	Convert string to all-uppercase
AnsiUpperCase (1384)	Convert string to all-uppercase
AppendStr (1385)	Append 2 strings
AssignStr (1386)	Assign value of strings on heap
CompareStr (1390)	Compare two strings case sensitive
CompareText (1391)	Compare two strings case insensitive
DisposeStr (1402)	Remove string from heap
IsValidIdent (1446)	Is string a valid pascal identifier
LastDelimiter (1446)	Last occurance of character in a string
LeftStr (1447)	Get first N characters of a string
LoadStr (1447)	Load string from resources
LowerCase (1448)	Convert string to all-lowercase
NewStr (1449)	Allocate new string on heap
RightStr (1452)	Get last N characters of a string
StrAlloc (1455)	Allocate memory for string
StrBufSize (1456)	Reserve memory for a string
StrDispose (1458)	Remove string from heap
StrPas (1467)	Convert PChar to pascal string
StrPCopy (1467)	Copy pascal string
StrPLCopy (1468)	Copy N bytes of pascal string
UpperCase (1487)	Convert string to all-uppercase

30.9 Used units

30.10 Overview

This documentation describes the sysutils unit. The sysutils unit was started by Gertjan Schouten, and completed by Michael Van Canneyt. It aims to be compatible to the Delphi sysutils unit, but in contrast with the latter, it is designed to work on multiple platforms. It is implemented on all supported platforms.

Table 30.7: Used units by unit 'sysutils'

Name	Page
errors	1350
sysconst	1350
Unix	1350
Unixtype	1350

30.11 Constants, types and variables

30.11.1 Constants

```
ConfigExtension : String = '.cfg'
```

`ConfigExtension` is the default extension used by the GetAppConfigFile (1436) call. It can be set to any valid extension for the current OS.

```
DateDelta = 693594
```

Days between 1/1/0001 and 12/31/1899

```
DriveDelim = DriveSeparator
```

`DriveDelim` refers to the system unit's `DriveSeparator` constant, it is for Delphi compatibility only.

```
EmptyStr : String = ''
```

Empty String Constant

```
EmptyWideStr : WideString = ''
```

Empty wide string.

```
faAnyFile = $0000003f
```

Use this attribute in the FindFirst (1419) call to find all matching files.

```
faArchive = $00000020
```

Attribute of a file, meaning the file has the archive bit set. Used in TSearchRec (1366) and FindFirst (1419)

```
faDirectory = $00000010
```

Attribute of a file, meaning the file is a directory. Used in TSearchRec (1366) and FindFirst (1419)

```
faHidden = $00000002
```

Attribute of a file, meaning the file is read-only. Used in TSearchRec (1366) and FindFirst (1419)

```
faReadOnly = $00000001
```

Attribute of a file, meaning the file is read-only. Used in TSearchRec (1366) and FindFirst (1419)

```
faSymLink = $00000040
```

faSymLink means the file (as returned e.g. by FindFirst (1419)/FindNext (1420)), is a symlink. It's ignored under Windows.

```
faSysFile = $00000004
```

Attribute of a file, meaning the file is a system file. Used in TSearchRec (1366) and FindFirst (1419)

```
faVolumeId = $00000008
```

Attribute of a file, meaning the file contains the volume ID. Used in TSearchRec (1366) and FindFirst (1419)

```
feInvalidHandle : THandle = THandle ( - 1 )
```

feInvalidHandle is the return value of FileOpen (1415) in case of an error.

```
filerecnamelength = 255
```

filerecnamelength describes the length of the FileRec (1362) filename field.

```
fmOpenRead = $0000
```

fmOpenRead is used in the FileOpen (1415) call to open a file in read-only mode.

```
fmOpenReadWrite = $0002
```

fmOpenReadWrite is used in the FileOpen (1415) call to open a file in read-write mode.

```
fmOpenWrite = $0001
```

fmOpenWrite is used in the FileOpen (1415) call to open a file in write-only mode.

```
fmShareCompat = $0000
```

fmOpenShareCompat is used in the FileOpen (1415) call OR-ed together with one of fmOpenReadWrite (1356), fmOpenRead (1356) or fmOpenWrite (1356),to open a file in a sharing modus that is equivalent to sharing implemented in MS-DOS.

```
fmShareDenyNone = $0040
```

fmOpenShareExclusive is used in the FileOpen (1415) call OR-ed together with one of fmOpenReadWrite (1356), fmOpenRead (1356) or fmOpenWrite (1356), to open a file so other processes can read/write the file as well.

```
fmShareDenyRead = $0030
```

fmOpenShareExclusive is used in the FileOpen (1415) call OR-ed together with one of fmOpenReadWrite (1356), fmOpenRead (1356) or fmOpenWrite (1356), to open a file so other processes cannot read from it.

```
fmShareDenyWrite = $0020
```

fmOpenShareExclusive is used in the FileOpen (1415) call OR-ed together with one of fmOpenReadWrite (1356), fmOpenRead (1356) or fmOpenWrite (1356), to open a file exclusively.

```
fmShareExclusive = $0010
```

fmOpenShareExclusive is used in the FileOpen (1415) call OR-ed together with one of fmOpenReadWrite (1356), fmOpenRead (1356) or fmOpenWrite (1356), to open a file exclusively.

```
fsFromBeginning = 0
```

fsFromBeginning is used to indicate in the FileSeek (1417) call that a seek operation should be started at the start of the file.

```
fsFromCurrent = 1
```

fsFromBeginning is used to indicate in the FileSeek (1417) call that a seek operation should be started at the current position in the file.

```
fsFromEnd = 2
```

fsFromBeginning is used to indicate in the FileSeek (1417) call that a seek operation should be started at the last position in the file.

```
HexDisplayPrefix : String = '$'
```

HexDisplayPrefix is used by the formatting routines to indicate that the number which follows the prefix is in Hexadecimal notation.

```
HoursPerDay = 24
```

Number of hours in a day.

```
JulianEpoch = TDateTime ( - 2415018.5 )
```

Starting point of the Julian calendar

```
LeadBytes : Set of Char = []
```

LeadBytes contains the set of bytes that serve as lead byte in a MBCS string.

```
MaxCurrency : Currency = 922337203685477.0000
```

Maximum currency value

```
MaxDateTime : TDateTime = 2958465.99999
```

Maximum TDateTime value.

```
MAX_PATH = MaxPathLen
```

MAX_PATH is the maximum number of characters that a filename (including path) can contain on the current operating system.

```
MinCurrency : Currency = -922337203685477.0000
```

Minimum Currency value

```
MinDateTime : TDateTime = -693593.0
```

Minimum TDateTime value.

```
MinsPerDay = HoursPerDay * MinsPerHour
```

Number of minutes per day.

```
MinsPerHour = 60
```

Number of minutes per hour.

```
MonthDays : Array[Boolean] of TDayTable = ( ( 31,28,31,30,31,30,31,31,30,31,30,31 )
```

Array with number of days in the months for leap and non-leap years.

```
MSecsPerDay = SecsPerDay * MSecsPerSec
```

Number of milliseconds per day

```
MSecsPerSec = 1000
```

Number of milliseconds per second

```
NullStr : PString = @EmptyStr
```

Pointer to an empty string

```
PathDelim = DirectorySeparator
```

PathDelim refers to the system unit's DirectorySeparator constant, it is for Delphi compatibility only.

```
PathSep = PathSeparator
```

PathSep refers to the system unit's PathSeparator constant, it is for Delphi compatibility only.

```
pfBCB4Produced = $08000000
```

Not used in Free Pascal.

```
pfDelphi4Produced = $0C000000
```

Not used in Free Pascal.

```
pfDesignOnly = $00000002
```

Package is a design-time only package

```
pfExeModule = $00000000
```

Package is an executable

```
pfIgnoreDupUnits = $00000008
```

Ignore duplicate units in package

```
pfLibraryModule = $80000000
```

Package is a library

```
pfModuleTypeMask = $C0000000
```

Mask for module type flags

```
pfNeverBuild = $00000001
```

Never-build flag was specified when compiling package

```
pfPackageModule = $40000000
```

Package is a real package (not exe)

```
pfProducerMask = $0C000000
```

Mask for producer flags

```
pfProducerUndefined = $04000000
```

Not used in Free Pascal.

```
pfRunOnly = $00000004
```

Package is a run-time only package

```
pfV3Produced = $00000000
```

Not used in Free Pascal.

```
RTL_SIGBUS = 4
```

Bus error signal number (Unix only)

```
RTL_SIGDEFAULT = -1
```

Default signal handler (Unix only)

```
RTL_SIGFPE = 1
```

Floating Point Error signal number (Unix only)

```
RTL_SIGILL = 3
```

Illegal instruction signal number (Unix only)

```
RTL_SIGINT = 0
```

INTERRUPT signal number (Unix only)

```
RTL_SIGLAST = RTL_SIGQUIT
```

Last signal number (Unix only)

```
RTL_SIGQUIT = 5
```

QUIT signal number (Unix only)

```
RTL_SIGSEGV = 2
```

Segmentation fault signal number (Unix only)

```
SecsPerDay = MinsPerDay * SecsPerMin
```

Number of seconds per day

```
SecsPerMin = 60
```

Number of seconds per minute

```
SwitchChars = ['-']
```

The characters in this set will be used by the FindCmdLineSwitch (1419) function to determine whether a command-line argument is a switch (an option) or a value. If the first character of an argument is in `SwitchChars`, it will be considered an option or switch.

```
SysConfigDir : String = ''
```

`SysConfigDir` is the default system configuration directory. It is set at application startup by the sysutils initialization routines.

This directory may be returned by the GetAppConfigDir (1436) call on some systems.

```
TextRecBufSize = 256
```

Buffer size of text file record.

```
TextRecNameLength = 256
```

Length of text file record filename field

```
ufImplicitUnit = $10
```

Unit was implicitly imported into package (did not appear in package contains list)

```
ufMainUnit = $01
```

Unit is the main unit of the package

```
ufOrgWeakUnit = $08
```

Unit is the original weak packaged unit

```
ufPackageUnit = $02
```

Unit is a packaged unit (appeared in package contains list)

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit
```

Weak (original or not) packaged unit

```
ufWeakUnit = $04
```

Unit is a weak packaged unit

```
UnixDateDelta = Trunc ( UnixEpoch )
```

Number of days between 1.1.1900 and 1.1.1970

```
UnixEpoch = JulianEpoch + TDateTime ( 2440587.5 )
```

Starting point of the unix calendar (1/1/1970)

30.11.2 Types

```
EHeapException = EHeapMemoryError
```

`EHeapMemoryError` is raised when an error occurs in the heap management routines.

```
ExceptClass = Class of Exception
```

`ExceptClass` is a Exception (1496) class reference.

```
FileRec = packed record
  Handle : THandle;
  Mode : LongInt;
  RecSize : SizeInt;
  _private : Array[1..3*SizeOf(SizeInt)+5*SizeOf(pointer)] of Byte;
  UserData : Array[1..32] of Byte;
  name : Array[0..filerecnamelength] of Char;
end
```

FileRec describes a untyped file. This record is made available so it can be used to implement drivers for other than the normal file system file records.

```
Int64Rec = packed record
end
```

Int64Rec can be used to extract the parts of a Int64: the high and low cardinal, or a zero-based array of 4 words, or a zero based array of 8 bytes. Note that the meaning of the High and Low parts are different on various CPUs.

```
LongRec = packed record
end
```

LongRec can be used to extract the parts of an long Integer: the high and low word, or the 4 separate bytes as a zero-based array of bytes. Note that the meaning of High and Low parts are different on various CPUs.

```
PByteArray = ^TByteArray
```

Generic pointer to TByteArray (1362). Use to access memory regions as a byte array.

```
PDayTable = ^TDayTable
```

Pointer to TDayTable type.

```
PString = ^String
```

Pointer to a ansistring

```
PSysCharSet = ^TSysCharSet
```

Pointer to TSysCharSet (1366) type.

```
PWordarray = ^TWordArray
```

Generic pointer to TWordArray (1367). Use to access memory regions as a word array.

```
TByteArray = Array[0..32767] of Byte
```

TByteArray is a generic array definition, mostly for use as a base type of the PByteArray (1362) type.

```
TCreateGUIDFunc = function(out GUID: TGUID) : Integer
```

TCreateGUIDFunc is the prototype for a GUID creation handler. On return, the GUID argument should contain a new (unique) GUID. The return value of the function should be zero for success, nonzero for failure.

```
TDayTable = Array[1..12] of Word
```

Array of day names.

```
TextBuf = Array[0..TextRecBufSize-1] of Char
```

TextBuf is the type for the default buffer in TextRec (1363)

```
TextRec = packed record
  Handle : THandle;
  Mode : LongInt;
  bufsize : SizeInt;
  _private : SizeInt;
  bufpos : SizeInt;
  bufend : SizeInt;
  bufptr : ^TextBuf;
  openfunc : pointer;
  inoutfunc : pointer;
  flushfunc : pointer;
  closefunc : pointer;
  UserData : Array[1..32] of Byte;
  name : Array[0..textrecnamelength-1] of Char;
  LineEnd : TLineEndStr;
  buffer : TextBuf;
end
```

TextRec describes a text file. This record is made available so it can be used to implement drivers for other than the normal file system file records.

To implement a driver, an Assign procedure must be implemented, which fills in the various fields of the record. Most notably, the callback functions must be filled in appropriately. After this, the normal file operations will handle all necessary calls to the various callbacks.

```
TFilename = String
```

TFileName is used in the TSearchRec (1366) definition.

```
TFileRec = FileRec
```

Alias for FileRec (1362) for Delphi compatibility.

```
TFloatFormat = (ffGeneral,ffExponent,ffFixed,ffNumber,ffCurrency)
```

Table 30.8: Enumeration values for type TFloatFormat

Value	Explanation
ffCurrency	Monetary format.
ffExponent	Scientific format.
ffFixed	Fixed point format.
ffGeneral	General number format.
ffNumber	Fixed point format with thousand separatord

TFloatFormat is used to determine how a float value should be formatted in the FloatToText (1425) function.

```
TFloatRec = record
  Exponent : Integer;
  Negative : Boolean;
  Digits : Array[0..18] of Char;
end
```

`TFloatRec` is used to descibe a floating point value by the FloatToDecimal (1421) function.

```
TFloatValue = (fvExtended,fvCurrency,fvSingle,fvReal,fvDouble,fvComp)
```

Table 30.9: Enumeration values for type TFloatValue

Value	Explanation
fvComp	Comp value
fvCurrency	Currency value
fvDouble	Double value
fvExtended	Extended value
fvReal	Real value
fvSingle	Single value

`TFloatValue` determines which kind of value should be returned in the (untyped) buffer used by the TextToFloat (1478) function.

```
TFormatSettings = record
  CurrencyFormat : Byte;
  NegCurrFormat : Byte;
  ThousandSeparator : Char;
  DecimalSeparator : Char;
  CurrencyDecimals : Byte;
  DateSeparator : Char;
  TimeSeparator : Char;
  ListSeparator : Char;
  CurrencyString : String;
  ShortDateFormat : String;
  LongDateFormat : String;
  TimeAMString : String;
  TimePMString : String;
  ShortTimeFormat : String;
  LongTimeFormat : String;
  ShortMonthNames : TMonthNameArray;
  LongMonthNames : TMonthNameArray;
  ShortDayNames : TWeekNameArray;
  LongDayNames : TWeekNameArray;
  TwoDigitYearCenturyWindow : Word;
end
```

`TFormatSettings` is a record that contains a copy of all variables which determine formatting in the various string formatting routines. It is used to pass local copies of these values to the various formatting routines in a thread-safe way.

```
TGetAppNameEvent = function : String
```

This callback type is used by the OnGetApplicationName (1369) to return an alternative application name.

```
TGetTempDirEvent = function(Global: Boolean) : String
```

Function prototype for OnGetTempDir (1369) handler.

```
TGetTempFileEvent = function(const Dir: String;const Prefix: String)
                                : String
```

Function prototype for OnGetTempFile (1369) handler.

```
TGetVendorNameEvent = function : String
```

`TGetVendorNameEvent` is the function prototype for the OnGetVendorName (1369) callback, used by the VendorName (1488) function.

```
THandle = System.THandle
```

`THandle` refers to the definition of `THandle` in the system unit, and is provided for backward compatibility only.

```
TIntegerSet = Set of
```

`TIntegerSet` is a generic integer subrange set definition whose size fits in a single integer.

```
TLineEndStr =
```

`TLineEndStr` is used in the TextRec (1363) record to indicate the end-of-line sequence for a text file.

```
TMbcsByteType = (mbSingleByte,mbLeadByte,mbTrailByte)
```

Table 30.10: Enumeration values for type TMbcsByteType

Value	Explanation
mbLeadByte	Uses lead-byte
mbSingleByte	Single bytes
mbTrailByte	Uses trailing byte

Type of multi-byte character set.

```
TMonthNameArray = Array[1..12] of String
```

`TMonthNameArray` is used in the month long and short name arrays.

```
TProcedure = procedure
```

TProcedure is a general definition of a procedural callback.

```
TReplaceFlags= Set of (rfReplaceAll,rfIgnoreCase)
```

TReplaceFlags determines the behaviour of the StringReplace (1461) function.

```
TSearchRec = record
  Time : LongInt;
  Size : Int64;
  Attr : LongInt;
  Name : TFilename;
  ExcludeAttr : LongInt;
  FindHandle : Pointer;
  Mode : TMode;
  PathOnly : AnsiString;
end
```

TSearchRec is a search handle description record. It is initialized by a call to FindFirst (1419) and can be used to do subsequent calls to FindNext (1420). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1419).

Remark: Not all fields of this record should be used. Some of the fields are for internal use only. (PathOnly for example, is only provided for Kylix compatibility)

```
TSignalState = (ssNotHooked,ssHooked,ssOverridden)
```

Table 30.11: Enumeration values for type TSignalState

Value	Explanation
ssHooked	A signal handler is set for the signal.
ssNotHooked	No signal handler is set for the signal.
ssOverridden	A signal handler was set for the signal

TSignalState indicates the state of a signal handler in a unix system for a particular signal.

```
TSysCharSet = Set of Char
```

Generic set of characters type.

```
TSysLocale = record
  DefaultLCID : Integer;
  PriLangID : Integer;
  SubLangID : Integer;
end
```

TSysLocale describes the current locale. If Fareast or MBCS is True, then the current locale uses a Multi-Byte Character Set. If MiddleEast or RightToLeft is True then words and sentences are read from right to left.

```
TTerminateProc = function : Boolean
```

TTerminateProc is the procedural type which should be used when adding exit procedures.

```
TTextRec = TextRec
```

Alias for TextRec (1363) for Delphi compatibility.

```
TTimeStamp = record
  Time : Integer;
  Date : Integer;
end
```

TTimeStamp contains a timestamp, with the date and time parts specified as separate TDateTime values.

```
TWeekNameArray = Array[1..7] of String
```

TWeekNameArray is used in the day long and short name arrays.

```
TWordArray = Array[0..16383] of Word
```

TWordArray is a generic array definition, mostly for use as a base type of the PWordArray (1362) type.

```
WordRec = packed record
  Lo : Byte;
  Hi : Byte;
end
```

LongRec can be used to extract the parts of a word: the high and low byte. Note that the meaning of the High and Low parts are different on various CPUs.

30.11.3 Variables

```
CurrencyDecimals : Byte
```

CurrencyDecimals is the number of decimals to be used when formatting a currency. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
CurrencyFormat : Byte
```

CurrencyFormat is the default format string for positive currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
CurrencyString : String
```

CurrencyString is the currency symbol for the current locale. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
DateSeparator : Char
```

DateSeparator is the character used by various date/time conversion routines as the character that separates the day from the month and the month from the year in a date notation. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
DecimalSeparator : Char
```

DecimalSeparator is used to display the decimal symbol in floating point numbers or currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
DefaultFormatSettings : TFormatSettings = (CurrencyFormat:1;NegCurrFormat:5;Thousand
```

DefaultFormatSettings contains the default settings for all type of formatting constants. If no thread-specific values are specified when a formatting function is called, this record is used as a default.

All other formatting constants refer to the fields of this variable using absolute addressing.

```
FalseBoolStrs : Array of String
```

FalseBoolStrs contains the strings that will result in a False return value by StrToBool (1469).

```
ListSeparator : Char
```

ListSeparator is the character used in lists of values. It is locale dependent.

```
LongDateFormat : String
```

LongDateFormat contains a template to format a date in a long format. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
LongDayNames : TWeekNameArray
```

LongDayNames is an array with the full names of days. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
LongMonthNames : TMonthNameArray
```

LongMonthNames is an array with the full names of months. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
LongTimeFormat : String
```

LongTimeFormat contains a template to format a time in full notation. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
NegCurrFormat : Byte
```

CurrencyFormat is the default format string for negative currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
OnCreateGUID : TCreateGUIDFunc = nil
```

OnCreateGUID can be set to point to a custom routine that creates GUID values. If set, the CreateGUID (1393) function will use it to obtain a GUID value. If it is not set, a default implementation using random values will be used to create the unique value. The function should return a valid GUID in the GUID parameter, and should return zero in case of success.

```
OnGetApplicationName : TGetAppNameEvent
```

By default, the configuration file routines GetAppConfigDir (1436) and GetAppConfigFile (1436) use a default application name to construct a directory or filename. This callback can be used to provide an alternative application name.

Since the result of this callback will be used to construct a filename, care should be taken that the returned name does not contain directory separator characters or characters that cannot appear in a filename.

```
OnGetTempDir : TGetTempDirEvent
```

OnGetTempDir can be used to provide custom behaviour for the GetTempDir (1440) function. Note that the returned name should have a trailing directory delimiter character.

```
OnGetTempFile : TGetTempFileEvent
```

OnGetTempDir can be used to provide custom behaviour for the GetTempFileName (1440) function. Note that the values for Prefix and Dir should be observed.

```
OnGetVendorName : TGetVendorNameEvent
```

OnGetVendorName is the callback used by the VendorName (1488) function. It should be set to a handler that returns the vendor of the application.

```
OnShowException : procedure(Msg: ShortString)
```

OnShowException is the callback that ShowException (1454) uses to display a message in a GUI application. For GUI applications, this variable should always be set. Note that no memory may be available when this callback is called, so the callback should already have all resources it needs, when the callback is set.

```
ShortDateFormat : String
```

ShortDateFormat contains a template to format a date in a short format. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
ShortDayNames : TWeekNameArray
```

ShortDayNames is an array with the abbreviated names of days. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
ShortMonthNames : TMonthNameArray
```

ShortMonthNames is an array with the abbreviated names of months. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
ShortTimeFormat : String
```

ShortTimeFormat contains a template to format a time in a short notation. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
SysLocale : TSysLocale
```

SysLocale is initialized by the initialization code of the SysUtils unit. For an explanation of the fields, see TSysLocale (1366)

```
ThousandSeparator : Char
```

ThousandSeparator is used to separate groups of thousands in floating point numbers or currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
TimeAMString : String
```

TimeAMString is used to display the AM symbol in the time formatting routines. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
TimePMString : String
```

TimePMString is used to display the PM symbol in the time formatting routines. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
TimeSeparator : Char
```

TimeSeparator is used by the time formatting routines to separate the hours from the minutes and the minutes from the seconds. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
TrueBoolStrs : Array of String
```

TrueBoolStrs contains the strings that will result in a True return value by StrToBool (1469).

```
TwoDigitYearCenturyWindow : Word
```

Window to determine what century 2 digit years are in.

30.12 Procedures and functions

30.12.1 AbandonSignalHandler

Synopsis:

Declaration: `procedure AbandonSignalHandler(RtlSigNum: Integer)`

Visibility: default

Description: This function is declared for Kylix compatibility, but is not implemented.

30.12.2 Abort

Synopsis: Abort program execution.

Declaration: `procedure Abort`

Visibility: default

Description: `Abort` raises an EAbort (1491) exception.

See also: Abort (1371)

30.12.3 AddDisk

Synopsis: Add a disk to the list of known disks (Unix only)

Declaration: `function AddDisk(const path: String) : Byte`

Visibility: default

Description: On Unix-like platforms both the DiskFree (1401) and DiskSize (1401) functions need a file on the specified drive, since is required for the statfs system call.

These filenames are set in drivestr[0..26], and the first 4 have been preset to :

Disk 0`'.'` default drive - hence current directory is used.

Disk 1`'/fd0/.'` floppy drive 1.

Disk 2`'/fd1/.'` floppy drive 2.

Disk 3`'/'` C: equivalent of DOS is the root partition.

Drives 4..26 can be set by your own applications with the `AddDisk` call.

The `AddDisk` call adds `Path` to the names of drive files, and returns the number of the disk that corresponds to this drive. If you add more than 21 drives, the count is wrapped to 4.

Errors: None.

See also: DiskFree (1401), DiskSize (1401)

30.12.4 AddTerminateProc

Synopsis: Add a procedure to the exit chain.

Declaration:
```
procedure AddTerminateProc(TermProc: TTerminateProc)
```

Visibility: default

Description: `AddTerminateProc` adds `TermProc` to the list of exit procedures. When the program exits, the list of exit procedures is run over, and all procedures are called one by one, in the reverse order that they were added to the exit chain.

Errors: If no memory is available on the heap, an exception may be raised.

See also: TTerminateProc (1367), CallTerminateProcs (1388)

30.12.5 AdjustLineBreaks

Synopsis: Convert possible line-endings to the currently valid line ending.

Declaration:
```
function AdjustLineBreaks(const S: String) : String
function AdjustLineBreaks(const S: String;Style: TTextLineBreakStyle)
                          : String
```

Visibility: default

Description: `AdjustLineBreaks` will change all `#13` characters with `#13#10` on Windowsnt and dos. On Unix-like platforms, all `#13#10` character pairs are converted to `#10` and single `#13` characters also.

Errors: None.

See also: AnsiCompareStr (1373), AnsiCompareText (1374)

Listing: ./sysutex/ex48.pp

```
Program Example48;

{ This program demonstrates the AdjustLineBreaks function }

Uses sysutils;

Const
  S = 'This is a string'#13'with embedded'#10'linefeed and'+
      #13'CR characters';

Begin
  WriteLn (AdjustLineBreaks(S));
End.
```

30.12.6 AnsiCompareFileName

Synopsis: Compare 2 filenames.

Declaration:
```
function AnsiCompareFileName(const S1: String;const S2: String)
                             : SizeInt
```

Visibility: default

Description: `AnsiCompareFileName` compares 2 filenames `S1` and `S2`, and returns

< 0if `S1<S2`.

= 0if `S1=S2`.

> 0if `S1>S2`.

The function actually checks `FileNameCaseSensitive` and returns the result of AnsiCompareStr (1373) or AnsiCompareText (1374) depending on whether `FileNameCaseSensitive` is `True` or `False`

Errors: None.

See also: AnsiCompareStr (1373), AnsiCompareText (1374), AnsiLowerCaseFileName (1377)

30.12.7 AnsiCompareStr

Synopsis: Compare 2 ansistrings, case sensitive, ignoring accents characters.

Declaration: `function AnsiCompareStr(const S1: String;const S2: String) : Integer`

Visibility: default

Description: `AnsiCompareStr` compares two strings and returns the following result:

< 0if `S1<S2`.

0if `S1=S2`.

> 0if `S1>S2`.

The comparision takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to AnsiCompareText (1374), the comparision is case sensitive.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AdjustLineBreaks (1372), AnsiCompareText (1374)

Listing: ./sysutex/ex49.pp

```
Program Example49;

{ This program demonstrates the AnsiCompareStr function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=AnsiCompareStr(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
```

```
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

30.12.8 AnsiCompareText

Synopsis: Compare 2 ansistrings, case insensitive, ignoring accents characters.

Declaration: `function AnsiCompareText(const S1: String;const S2: String) : Integer`

Visibility: default

Description: `AnsiCompareText` compares two strings and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

the comparision takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to AnsiCompareStr (1373), the comparision is case insensitive.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AdjustLineBreaks (1372), AnsiCompareText (1374)

Listing: ./sysutex/ex50.pp

```
Program Example49;

{ This program demonstrates the AnsiCompareText function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=AnsiCompareText(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
```

```
    Write ('larger than ');
  WriteLn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

30.12.9 AnsiDequotedStr

Synopsis:

Declaration: `function AnsiDequotedStr(const S: String;AQuote: Char) : String`

Visibility: default

Description:

Errors:

30.12.10 AnsiExtractQuotedStr

Synopsis: Removes the first quoted string from a string.

Declaration: `function AnsiExtractQuotedStr(var Src: PChar;Quote: Char) : String`

Visibility: default

Description: `AnsiExtractQuotedStr` returns the first quoted string in `Src`, and deletes the result from `Src`. The resulting string has with `Quote` characters removed from the beginning and end of the string (if they are present), and double `Quote` characters replaced by a single `Quote` characters. As such, it revereses the action of AnsiQuotedStr (1377).

Errors: None.

See also: AnsiQuotedStr (1377)

Listing: ./sysutex/ex51.pp

```
Program Example51;

{ This program demonstrates the AnsiQuotedStr function }

Uses sysutils;

Var S : AnsiString;

Begin
  S:='He said "Hello" and walked on';
  S:=AnsiQuotedStr(Pchar(S),'"');
  WriteLn (S);
  WriteLn(AnsiExtractQuotedStr(Pchar(S),'"'));
End.
```

30.12.11 AnsiLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiLastChar(const S: String) : PChar`

Visibility: default

Description: This function returns a pointer to the last character of `S`.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets. If none is installed, this function is the same as `@S[Length[S]]`.

Errors: None.

See also: AnsiStrLastChar (1380)

Listing: ./sysutex/ex52.pp

```
Program Example52;

{ This program demonstrates the AnsiLastChar function }

Uses sysutils;

Var S : AnsiString;
    L : Longint;

Begin
  S:='This is an ansistring.';
  Writeln ('Last character of S is : ',AnsiLastChar(S));
  L:=Longint(AnsiLastChar(S))-Longint(@S[1])+1;
  Writeln ('Length of S is : ',L);
End.
```

30.12.12 AnsiLowerCase

Synopsis: Return a lowercase version of a string.

Declaration: `function AnsiLowerCase(const s: String) : String`

Visibility: default

Description: `AnsiLowerCase` converts the string `S` to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so spcial characters are converted correctly as well.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiUpperCase (1384), AnsiStrLower (1382), AnsiStrUpper (1384)

Listing: ./sysutex/ex53.pp

```
Program Example53;

{ This program demonstrates the AnsiLowerCase function }
```

```
Uses sysutils;

Procedure Testit (S : String);

begin
 WriteIn (S,' -> ',AnsiLowerCase(S))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

30.12.13 AnsiLowerCaseFileName

Synopsis: Convert filename to lowercase.

Declaration: `function AnsiLowerCaseFileName(const s: String) : String`

Visibility: default

Description: `AnsiLowerCaseFileName` simply returns the result of

```
AnsiLowerCase(S);
```

See also: AnsiLowerCase (1376), AnsiCompareFileName (1372), AnsiUpperCaseFileName (1385)

30.12.14 AnsiPos

Synopsis: Return Position of one ansistring in another.

Declaration: `function AnsiPos(const substr: String;const s: String) : SizeInt`

Visibility: default

Description: `AnsiPos` does the same as the standard `Pos` function.

See also: AnsiStrPos (1383), AnsiStrScan (1383), AnsiStrRScan (1383)

30.12.15 AnsiQuotedStr

Synopsis: Return a quoted version of a string.

Declaration: `function AnsiQuotedStr(const S: String;Quote: Char) : String`

Visibility: default

Description: `AnsiQuotedString` quotes the string `S` and returns the result. This means that it puts the `Quote` character at both the beginning and end of the string and replaces any occurrence of `Quote` in `S` with 2 `Quote` characters. The action of `AnsiQuotedString` can be reversed by AnsiExtractQuotedStr (1375).

For an example, see AnsiExtractQuotedStr (1375)

Errors: None.

See also: AnsiExtractQuotedStr (1375)

30.12.16 AnsiSameStr

Synopsis: Checks whether 2 strings are the same (case sensitive)

Declaration: `function AnsiSameStr(const s1: String;const s2: String) : Boolean`

Visibility: default

Description: `SameText` calls AnsiCompareStr (1373) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareStr (1373), SameText (1453), AnsiSameText (1378)

30.12.17 AnsiSameText

Synopsis: Checks whether 2 strings are the same (case insensitive)

Declaration: `function AnsiSameText(const s1: String;const s2: String) : Boolean`

Visibility: default

Description: `SameText` calls AnsiCompareText (1374) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

Errors:

See also: AnsiCompareText (1374), SameText (1453), AnsiSameStr (1378)

30.12.18 AnsiStrComp

Synopsis: Compare two null-terminated strings. Case sensitive.

Declaration: `function AnsiStrComp(S1: PChar;S2: PChar) : Integer`

Visibility: default

Description: `AnsiStrComp` compares 2 `PChar` strings, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparision of the two strings is case-sensitive.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareText (1374), AnsiCompareStr (1373)

Listing: ./sysutex/ex54.pp

```
Program Example54;

{ This program demonstrates the AnsiStrComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin
  R:=AnsiStrComp(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

30.12.19 AnsiStrIComp

Synopsis: Compare two null-terminated strings. Case insensitive.

Declaration: `function AnsiStrIComp(S1: PChar;S2: PChar) : Integer`

Visibility: default

Description: `AnsiStrIComp` compares 2 `PChar` strings, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparision of the two strings is case-insensitive.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareText (1374), AnsiCompareStr (1373)

Listing: ./sysutex/ex55.pp

```
Program Example55;

{ This program demonstrates the AnsiStrIComp function }
```

```
Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin
  R:=AnsiStrIComp(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  Writeln ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

30.12.20 AnsiStrLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiStrLastChar(Str: PChar) : PChar`

Visibility: default

Description: Return a pointer to the last character of the null-terminated string.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets. If none is installed, this function is the same as `@S[Length[S]]`.

Errors: None.

See also: AnsiCompareText (1374), AnsiCompareStr (1373)

Listing: ./sysutex/ex56.pp

```
Program Example56;

{ This program demonstrates the AnsiStrLComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin
  R:=AnsiStrLComp(S1,S2,L);
  Write ('First ',L,' characters of "',S1,'" are ');
```

```
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('those of "',S2,'"');
end;

Begin
  Testit('One string','One smaller string',255);
  Testit('One string','One String',4);
  Testit('One string','1 string',0);
  Testit('One string','One string.',9);
End.
```

30.12.21 AnsiStrLComp

Synopsis: Compare a limited number of characters of 2 strings

Declaration: `function AnsiStrLComp(S1: PChar;S2: PChar;MaxLen: cardinal) : Integer`

Visibility: default

Description: `AnsiStrLComp` functions the same as AnsiStrComp (1378), but compares at most `MaxLen` characters. If the first `MaxLen` characters in both strings are the same, then zero is returned.

Note that this function processes embedded null characters, treating them as a normal character.

Errors: None.

See also: AnsiStrComp (1378), AnsiStrIComp (1379), AnsiStrLIComp (1381)

30.12.22 AnsiStrLIComp

Synopsis: Compares a given number of characters of a string, case insensitive.

Declaration: `function AnsiStrLIComp(S1: PChar;S2: PChar;MaxLen: cardinal) : Integer`

Visibility: default

Description: `AnsiStrLIComp` compares the first `Maxlen` characters of 2 `PChar` strings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparision of the two strings is case-insensitive.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareText (1374), AnsiCompareStr (1373)

Listing: ./sysutex/ex57.pp

```
Program Example57;

{ This program demonstrates the AnsiStrLIComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin
  R:=AnsiStrLIComp(S1,S2,L);
  Write ('First ',L,' characters of "',S1,'" are ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('those of "',S2,'"');
end;

Begin
  Testit('One string','One smaller string',255);
  Testit('ONE STRING','one String',4);
  Testit('One string','1 STRING',0);
  Testit('One STRING','one string.',9);
End.
```

30.12.23 AnsiStrLower

Synopsis: Convert a null-terminated string to all-lowercase characters.

Declaration: `function AnsiStrLower(Str: PChar) : PChar`

Visibility: default

Description: `AnsiStrLower` converts the PChar `Str` to lowercase characters and returns the resulting pchar. Note that `Str` itself is modified, not a copy, as in the case of AnsiLowerCase (1376). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiStrUpper (1384), AnsiLowerCase (1376)

Listing: ./sysutex/ex59.pp

```
Program Example59;

{ This program demonstrates the AnsiStrLower function }

Uses sysutils;
```

```
Procedure Testit (S : Pchar);

begin
 WriteIn (S,' -> ',AnsiStrLower(S))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

30.12.24 AnsiStrPos

Synopsis: Return position of one null-terminated substring in another

Declaration: `function AnsiStrPos(str: PChar;substr: PChar) : PChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the first occurrence of `SubStr` in `Str`. If `SubStr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if either `Str` or `SubStr` point to invalid memory.

See also: AnsiPos (1377), AnsiStrScan (1383), AnsiStrRScan (1383)

30.12.25 AnsiStrRScan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrRScan(Str: PChar;Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the *last* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: AnsiPos (1377), AnsiStrScan (1383), AnsiStrPos (1383)

30.12.26 AnsiStrScan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrScan(Str: PChar;Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the *first* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: AnsiPos (1377), AnsiStrScan (1383), AnsiStrPos (1383)

30.12.27 AnsiStrUpper

Synopsis: Convert a null-terminated string to all-uppercase characters.

Declaration: `function AnsiStrUpper(Str: PChar) : PChar`

Visibility: default

Description: `AnsiStrUpper` converts the `PCharStr` to uppercase characters and returns the resulting string. Note that `Str` itself is modified, not a copy, as in the case of AnsiUpperCase (1384). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiUpperCase (1384), AnsiStrLower (1382), AnsiLowerCase (1376)

Listing: ./sysutex/ex60.pp

```
Program Example60;

{ This program demonstrates the AnsiStrUpper function }

Uses sysutils;

Procedure Testit (S : Pchar);

begin
  WriteIn (S,' -> ',AnsiStrUpper(S))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

30.12.28 AnsiUpperCase

Synopsis: Return an uppercase version of a string, taking into account special characters.

Declaration: `function AnsiUpperCase(const s: String) : String`

Visibility: default

Description: `AnsiUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark: A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiStrUpper (1384), AnsiStrLower (1382), AnsiLowerCase (1376)

Listing: ./sysutex/ex61.pp

```
Program Example60;

{ This program demonstrates the AnsiUpperCase function }

Uses sysutils;

Procedure Testit (S : String);

begin
 WriteIn (S,' -> ',AnsiUpperCase(S))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

30.12.29 AnsiUpperCaseFileName

Synopsis: Convert filename to uppercase.

Declaration: `function AnsiUpperCaseFileName(const s: String) : String`

Visibility: default

Description: `AnsiUpperCaseFileName` simply returns the result of

```
AnsiUpperCase(S);
```

See also: AnsiUpperCase (1384), AnsiCompareFileName (1372), AnsiLowerCaseFileName (1377)

30.12.30 AppendStr

Synopsis: Append one ansistring to another.

Declaration: `procedure AppendStr(var Dest: String;const S: String)`

Visibility: default

Description: `AppendStr` appends `S` to Dest.

This function is provided for Delphi compatibility only, since it is completely equivalent to `Dest:=Dest+S`.

Errors: None.

See also: AssignStr (1386), NewStr (1449), DisposeStr (1402)

Listing: ./sysutex/ex62.pp

```
Program Example62;

{ This program demonstrates the AppendStr function }

Uses sysutils;

Var S : AnsiString;
```

```
Begin
  S:='This is an ';
  AppendStr(S,'AnsiString');
  Writeln ('S = "',S,'"');
End.
```

30.12.31 ApplicationName

Synopsis: Return a default application name

Declaration: `function ApplicationName : String`

Visibility: default

Description: `ApplicationName` returns the name of the current application. Standard this is equal to the result of `ParamStr(0)`, but it can be customized by setting the OnGetApplicationName (1369) callback.

Errors: None.

See also: GetAppConfigDir (1436), OnGetApplicationName (1369), GetAppConfigFile (1436), ConfigExtension (1355)

30.12.32 AssignStr

Synopsis: Assigns an ansistring to a null-terminated string.

Declaration: `procedure AssignStr(var P: PString;const S: String)`

Visibility: default

Description: `AssignStr` allocates `S` to `P`. The old value of `P` is disposed of.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

Errors: None.

See also: NewStr (1449), AppendStr (1385), DisposeStr (1402)

Listing: ./sysutex/ex63.pp

```
Program Example63;

{ This program demonstrates the AssignStr function }
{$H+}

Uses sysutils;

Var P : PString;

Begin
 P:=NewStr('A first AnsiString');
 Writeln ('Before: P = "',P^,'"');
 AssignStr(P,'A Second ansistring');
 Writeln ('After : P = "',P^,'"');
 DisposeStr(P);
End.
```

30.12.33 BCDToInt

Synopsis: Convert a BCD coded integer to a normal integer.

Declaration: `function BCDToInt(Value: Integer) : Integer`

Visibility: default

Description: `BCDToInt` converts a `BCD` coded integer to a normal integer.

Errors: None.

See also: StrToInt (1474), IntToStr (1444)

Listing: ./sysutex/ex64.pp

```
Program Example64;

{ This program demonstrates the BCDToInt function }

Uses sysutils;

Procedure Testit ( L : longint);
begin
  WriteIn (L,' -> ',BCDToInt(L));
end;

Begin
  Testit(10);
  Testit(100);
  Testit(1000);
End.
```

30.12.34 Beep

Synopsis: Sound the system bell.

Declaration: `procedure Beep`

Visibility: default

Description: `Beep` sounds the system bell, if one is available.

Errors: This routine may not be implemented on all platforms.

30.12.35 BoolToStr

Synopsis: Convert a boolean value to a string.

Declaration: `function BoolToStr(B: Boolean;UseBoolStrs: Boolean) : String`

Visibility: default

Description: `BoolToStr` converts the boolean `B` to one of the strings `'TRUE'` or `'FALSE'`

Errors: None.

See also: StrToBool (1469)

30.12.36 ByteToCharIndex

Synopsis: Convert a character index in Bytes to an Index in characters

Declaration: `function ByteToCharIndex(const S: String;Index: Integer) : Integer`

Visibility: default

Description: `ByteToCharIndex` returns the index (in characters) of the `Index`-th byte in `S`.

Errors: This function does not take into account MBCS yet.

See also: CharToByteLen (1389), ByteToCharLen (1388)

30.12.37 ByteToCharLen

Synopsis: Convert a length in bytes to a length in characters.

Declaration: `function ByteToCharLen(const S: String;MaxLen: Integer) : Integer`

Visibility: default

Description: `ByteToCharLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: CharToByteLen (1389), ByteToCharIndex (1388)

30.12.38 ByteType

Synopsis: Return the type of byte in an ansistring for a multi-byte character set

Declaration: `function ByteType(const S: String;Index: Integer) : TMbcsByteType`

Visibility: default

Description: `ByteType` returns the type of byte in the ansistring `S` at (1-based) position `Index`.

Errors: No checking on the index is performed.

See also: TMbcsByteType (1365), StrByteType (1456)

30.12.39 CallTerminateProcs

Synopsis: Call the exit chain procedures.

Declaration: `function CallTerminateProcs : Boolean`

Visibility: default

Description: `CallTerminateProcs` is run on program exit. It executes all terminate procedures that were added to the exit chain with AddTerminateProc (1372), and does this in reverse order.

Errors: If one of the exit procedure raises an exception, it is *not* caught, and the remaining exit procedures will not be executed.

See also: TTerminateProc (1367), AddTerminateProc (1372)

30.12.40 ChangeFileExt

Synopsis: Change the extension of a filename.

Declaration:
```
function ChangeFileExt(const FileName: String;const Extension: String)
                       : String
```

Visibility: default

Description: `ChangeFileExt` changes the file extension in `FileName` to `Extension`. The extension `Extension` includes the starting `.` (dot). The previous extension of `FileName` are all characters after the last `.`, the `.` character included.

If `FileName` doesn't have an extension, `Extension` is just appended.

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExpandFileName (1406)

30.12.41 CharToByteLen

Synopsis: Convert a length in characters to a length in bytes.

Declaration:
```
function CharToByteLen(const S: String;MaxLen: Integer) : Integer
```

Visibility: default

Description: `CharToByteLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: ByteToCharLen (1388), ByteToCharIndex (1388)

30.12.42 CompareMem

Synopsis: Compare two memory areas.

Declaration:
```
function CompareMem(P1: Pointer;P2: Pointer;Length: cardinal) : Boolean
```

Visibility: default

Description: `CompareMem` compares, byte by byte, 2 memory areas pointed to by `P1` and `P2`, for a length of `L` bytes.

It returns the following values:

<0if at some position the byte at `P1` is less than the byte at the same postion at `P2`.

0if all `L` bytes are the same.

>0if at some position the byte at `P1` is greater than the byte at the same postion at `P2`.

Errors:

30.12.43 CompareMemRange

Synopsis: Compare 2 memory locations

Declaration:
```
function CompareMemRange(P1: Pointer;P2: Pointer;Length: cardinal)
                         : Integer
```

Visibility: default

Description: `CompareMemRange` compares the 2 memory locations pointed to by `P1` and `P2` byte per byte. It stops comparing after `Length` bytes have been compared, or when it has encountered 2 different bytes. The result is then

>0 if a byte in range `P1` was found that is bigger than the corresponding byte in range `P2`.

0 if all bytes in range `P1` are the same as the corresponding bytes in range `P2`.

<0 if a byte in range `P1` was found that is less than the corresponding byte in range `P2`.

Errors: None.

See also: SameText (1453)

30.12.44 CompareStr

Synopsis: Compare 2 ansistrings case-sensitively, ignoring special characters.

Declaration:
```
function CompareStr(const S1: String;const S2: String) : Integer
                    ;  Overload
```

Visibility: default

Description: `CompareStr` compares two strings, `S1` and `S2`, and returns the following result:

<0 if `S1<S2`.

0 if `S1=S2`.

>0 if `S1>S2`.

The comparision of the two strings is case-sensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: AnsiCompareText (1374), AnsiCompareStr (1373), CompareText (1391)

Listing: ./sysutex/ex65.pp

```
Program Example65;

{ This program demonstrates the CompareStr function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=CompareStr(S1,S2);
```

```
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  Writeln ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

30.12.45 CompareText

Synopsis: Compare 2 ansistrings case insensitive.

Declaration: `function CompareText(const S1: String;const S2: String) : Integer`

Visibility: default

Description: `CompareText` compares two strings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparision of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: AnsiCompareText (1374), AnsiCompareStr (1373), CompareStr (1390)

Listing: ./sysutex/ex66.pp

```
Program Example66;

{ This program demonstrates the CompareText function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=CompareText(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
```

```
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteLn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

30.12.46 ComposeDateTime

Synopsis: Add a date and time

Declaration: `function ComposeDateTime(Date: TDateTime;Time: TDateTime) : TDateTime`

Visibility: default

Description: `ComposeDateTime` correctly adds `Date` and `Time`, also for dates before 1899-12-31. For dates after this date, it is just the mathematical addition.

Errors: None.

See also: EncodeDateTime (1350)

30.12.47 CreateDir

Synopsis: Create a new directory

Declaration: `function CreateDir(const NewDir: String) : Boolean`

Visibility: default

Description: `CreateDir` creates a new directory with name `NewDir`. If the directory doesn't contain an absolute path, then the directory is created below the current working directory.

The function returns `True` if the directory was successfully created, `False` otherwise.

Errors: In case of an error, the function returns `False`.

See also: RemoveDir (1451)

Listing: ./sysutex/ex26.pp

```
Program Example26;

{ This program demonstrates the CreateDir and RemoveDir functions }
{ Run this program twice in the same directory }

Uses sysutils;

Begin
  If Not DirectoryExists('NewDir') then
    If Not CreateDir ('NewDir') Then
      WriteLn ('Failed to create directory !')
```

```
    else
      WriteIn ('Created "NewDir" directory')
  Else
    If Not RemoveDir ('NewDir') Then
      WriteIn ('Failed to remove directory !')
    else
      WriteIn ('Removed "NewDir" directory');
End.
```

30.12.48 CreateGUID

Synopsis: Create a new GUID

Declaration: `function CreateGUID(out GUID: TGUID) : Integer`

Visibility: default

Description: `CreateGUID` can be called to create a new GUID (Globally Unique Identifier) value. The function returns the new `GUID` value in `GUID` and returns zero in case the GUID was created succesfully. If no GUID was created, a nonzero error code is returned.

The default mechanism for creating a new GUID is system dependent. If operating system support is available, it is used. If none is available, a default implementation using random numbers is used.

The `OnCreateGUID` callback can be set to hook a custom mechanism behind the `CreateGUID` function. This can be used to lt the GUID be created by an external GUID creation library.

Errors: On error, a nonzero return value is returned.

See also: CreateGUID (1393)

30.12.49 CurrentYear

Synopsis: Return the current year

Declaration: `function CurrentYear : Word`

Visibility: default

Description: `CurrentYear` returns the current year as a 4-digit number.

Errors: None.

See also: Date (1394), Time (1479), Now (1449)

30.12.50 CurrToStr

Synopsis: Convert a currency value to a string.

Declaration: `function CurrToStr(Value: Currency) : String`

Visibility: default

Description: `CurrToStr` will convert a currency value to a string with a maximum of 15 digits, and precision 2. Calling `CurrToStr` is equivalent to calling FloatToStrF (1423):

```
FloatToStrF(Value,ffNumber,15,2);
```

Errors: None.

See also: FloatToStrF (1423), StrToCurr (1470)

30.12.51 CurrToStrF

Synopsis: Format a currency to a string

Declaration:
```
function CurrToStrF(Value: Currency;Format: TFloatFormat;
                    Digits: Integer) : String
function CurrToStrF(Value: Currency;Format: TFloatFormat;
                    Digits: Integer;
                    const FormatSettings: TFormatSettings) : String
```

Visibility: default

Description: `CurrToStrF` formats the currency `Value` according to the value in `Format`, using the number of digits specified in `Digits`, and a precision of 19. This function simply calls FloatToStrF (1423).

See also: FloatToStrF (1423)

30.12.52 Date

Synopsis: Return the current date.

Declaration:
```
function Date : TDateTime
```

Visibility: default

Description: `Date` returns the current date in `TDateTime` format.

Errors: None.

See also: Time (1479), Now (1449)

Listing: ./sysutex/ex1.pp

```
Program Example1;

{ This program demonstrates the Date function }

uses sysutils;

Var YY,MM,DD : Word;

Begin
 WriteLn ('Date : ',Date);
 DeCodeDate (Date,YY,MM,DD);
 WriteLn (format ('Date is (DD/MM/YY): %d/%d/%d ',[dd,mm,yy]));
End.
```

30.12.53 DateTimeToFileDate

Synopsis: Convert a `TDateTime` value to a file age (integer)

Declaration:
```
function DateTimeToFileDate(DateTime: TDateTime) : LongInt
```

Visibility: default

Description: `DateTimeToFileDate` function converts a date/time indication in `TDateTime` format to a file-date function, such as returned for instance by the FileAge (1410) function.

Errors: None.

See also: Time (1479), Date (1394), FileDateToDateTime (1412), DateTimeToSystemTime (1396), DateTimeToTimeStamp (1397)

Listing: ./sysutex/ex2.pp

```
Program Example2;

{ This program demonstrates the DateTimeToFileDate function }

Uses sysutils;

Begin
  WriteLn ('FileTime of now would be: ',DateTimeToFileDate (Now));
End.
```

30.12.54 DateTimeToStr

Synopsis: Converts a `TDateTime` value to a string using a predefined format.

Declaration: `function DateTimeToStr(DateTime: TDateTime) : String`

Visibility: default

Description: `DateTimeToStr` returns a string representation of `DateTime` using the formatting specified in `LongDateTimeFormat`. It corresponds to a call to `FormatDateTime('c',DateTime)` (see formatchars (1353)).

Errors: None.

See also: FormatDateTime (1433)

Listing: ./sysutex/ex3.pp

```
Program Example3;

{ This program demonstrates the DateTimeToStr function }

Uses sysutils;

Begin
  WriteLn ('Today is : ',DateTimeToStr(Now));
  WriteLn ('Today is : ',FormatDateTime('c',Now));
End.
```

30.12.55 DateTimeToString

Synopsis: Converts a `TDateTime` value to a string with a given format.

Declaration: `procedure DateTimeToString(out Result: String;const FormatStr: String; const DateTime: TDateTime)`

Visibility: default

Description: `DateTimeToString` returns in `Result` a string representation of `DateTime` using the formatting specified in `FormatStr`. for a list of characters that can be used in the `FormatStr` formatting string, see formatchars (1353).

Errors: In case a wrong formatting character is found, an `EConvertError` is raised.

See also: FormatDateTime (1433), formatchars (1353)

Listing: ./sysutex/ex4.pp

```
Program Example4;

{ This program demonstrates the DateTimeToString function }

Uses sysutils;


Procedure today (Fmt : string);

Var S : AnsiString;

begin
  DateTimeToString (S,Fmt,Date);
  Writeln (S);
end;

Procedure Now (Fmt : string);

Var S : AnsiString;

begin
  DateTimeToString (S,Fmt,Time);
  Writeln (S);
end;

Begin
  Today ('"Today is "dddd dd mmmm y');
  Today ('"Today is "d mmm yy');
  Today ('"Today is "d/mmm/yy');
  Now ('''The time is ''am/pmh:n:s');
  Now ('''The time is ''hh:nn:ssam/pm');
  Now ('''The time is ''tt');
End.
```

30.12.56 DateTimeToSystemTime

Synopsis: Converts a `TDateTime` value to a systemtime structure.

Declaration:
```
procedure DateTimeToSystemTime(DateTime: TDateTime;
                               out SystemTime: TSystemTime)
```

Visibility: default

Description: `DateTimeToSystemTime` converts a date/time pair in `DateTime`, with `TDateTime` format to a system time `SystemTime`.

Errors: None.

See also: DateTimeToFileDate (1394), SystemTimeToDateTime (1478), DateTimeToTimeStamp (1397)

Listing: ./sysutex/ex5.pp

```
Program Example5;

{ This program demonstrates the DateTimeToSystemTime function }

Uses sysutils;

Var ST : TSystemTime;

Begin
  DateTimeToSystemTime(Now,ST);
  With St do
    begin
    Writeln ('Today is     ',year,'/',month,'/',Day);
    Writeln ('The time is ',Hour,':',minute,':',Second,'.',MilliSecond);
    end;
End.
```

30.12.57 DateTimeToTimeStamp

Synopsis: Converts a `TDateTime` value to a TimeStamp structure.

Declaration: `function DateTimeToTimeStamp(DateTime: TDateTime) : TTimeStamp`

Visibility: default

Description: `DateTimeToSystemTime` converts a date/time pair in `DateTime`, with `TDateTime` format to a `TTimeStamp` format.

Errors: None.

See also: DateTimeToFileDate (1394), SystemTimeToDateTime (1478), DateTimeToSystemTime (1396)

Listing: ./sysutex/ex6.pp

```
Program Example6;

{ This program demonstrates the DateTimeToTimeStamp function }

Uses sysutils;

Var TS : TTimeStamp;

Begin
  TS:=DateTimeToTimeStamp (Now);
  With TS do
    begin
    Writeln ('Now is ',time,' millisecond past midnight');
    Writeln ('Today is ' ,Date,' days past 1/1/0001');
    end;
End.
```

30.12.58 DateToStr

Synopsis: Converts a `TDateTime` value to a date string with a predefined format.

Declaration: `function DateToStr(Date: TDateTime) : String`

Visibility: default

Description: `DateToStr` converts `Date` to a string representation. It uses `ShortDateFormat` as it's formatting string. It is hence completely equivalent to a `FormatDateTime('ddddd', Date)`.

Errors: None.

See also: TimeToStr (1481), DateTimeToStr (1395), FormatDateTime (1433), StrToDate (1470)

Listing: ./sysutex/ex7.pp

```
Program Example7;

{ This program demonstrates the DateToStr function }

Uses sysutils;

Begin
  WriteLn(Format ('Today is: %s',[DateToStr(Date)]));
End.
```

30.12.59 DayOfWeek

Synopsis: Returns the day of the week.

Declaration: `function DayOfWeek(DateTime: TDateTime) : Integer`

Visibility: default

Description: `DayOfWeek` returns the day of the week from `DateTime`. `Sunday` is counted as day 1, `Saturday` is counted as day 7. The result of `DayOfWeek` can serve as an index to the `LongDayNames` constant array, to retrieve the name of the day.

Errors: None.

See also: Date (1394), DateToStr (1397)

Listing: ./sysutex/ex8.pp

```
Program Example8;

{ This program demonstrates the DayOfWeek function }

Uses sysutils;

Begin
  WriteLn ('Today''s day is ',LongDayNames[DayOfWeek(Date)]);
End.
```

30.12.60 DecodeDate

Synopsis: Decode a `TDateTime` to a year,month,day triplet

Declaration: `procedure DecodeDate(Date: TDateTime;out Year: Word;out Month: Word; out Day: Word)`

Visibility: default

Description: `DecodeDate` decodes the Year, Month and Day stored in `Date`, and returns them in the `Year`, `Month` and `Day` variables.

Errors: None.

See also: EncodeDate (1403), DecodeTime (1399)

Listing: ./sysutex/ex9.pp

```
Program Example9;

{ This program demonstrates the DecodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin
  DecodeDate(Date,YY,MM,DD);
  Writeln (Format ('Today is %d/%d/%d',[dd,mm,yy]));
End.
```

30.12.61 DecodeDateFully

Synopsis: Decode a date with additional date of the week.

Declaration:
```
function DecodeDateFully(const DateTime: TDateTime;out Year: Word;
                         out Month: Word;out Day: Word;out DOW: Word)
                          : Boolean
```

Visibility: default

Description: `DecodeDateFully`, like DecodeDate (1398), decodes `DateTime` in its parts and returns these in `Year`, `Month`, `Day` but in addition returns the day of the week in `DOW`.

Errors: None.

See also: EncodeDate (1403), TryEncodeDate (1483), DecodeDate (1398)

30.12.62 DecodeTime

Synopsis: Decode a `TDateTime` to a hour,minute,second,millisec quartet

Declaration:
```
procedure DecodeTime(Time: TDateTime;out Hour: Word;out Minute: Word;
                     out Second: Word;out MilliSecond: Word)
```

Visibility: default

Description: `DecodeDate` decodes the hours, minutes, second and milliseconds stored in `Time`, and returns them in the `Hour`, `Minute` and `Second` and `MilliSecond` variables.

Errors: None.

See also: EncodeTime (1404), DecodeDate (1398)

Listing: ./sysutex/ex10.pp

```
Program Example10;

{ This program demonstrates the DecodeTime function }

Uses sysutils;

Var HH,MM,SS,MS: Word;

Begin
  DecodeTime(Time,HH,MM,SS,MS);
  Writeln (format('The time is %d:%d:%d.%d',[hh,mm,ss,ms]));
End.
```

30.12.63 DeleteFile

Synopsis: Delete a file from the filesystem.

Declaration: `function DeleteFile(const FileName: String) : Boolean`

Visibility: default

Description: `DeleteFile` deletes file `FileName` from disk. The function returns `True` if the file was successfully removed, `False` otherwise.

Errors: On error, `False` is returned.

See also: FileCreate (1411), FileExists (1413)

Listing: ./sysutex/ex31.pp

```
Program Example31;

{ This program demonstrates the DeleteFile function }

Uses sysutils;

Var
  Line : String;
  F,I : Longint;

Begin
  F:=FileCreate('test.txt');
  Line:='Some string line.'#10;
  For I:=1 to 10 do
    FileWrite (F,Line[1],Length(Line));
  FileClose(F);
  DeleteFile('test.txt');
End.
```

30.12.64 DirectoryExists

Synopsis: Check whether a directory exists in the file system.

Declaration: `function DirectoryExists(const Directory: String) : Boolean`

Visibility: default

Description: `DirectoryExists` checks whether `Directory` exists in the filesystem and is actually a directory. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: FileExists (1413)

30.12.65 DiskFree

Synopsis: Return the amount of free diskspace

Declaration: `function DiskFree(drive: Byte) : Int64`

Visibility: default

Description: `DiskFree` returns the free space (in bytes) on disk `Drive`. Drive is the number of the disk drive:

0for the current drive.

1for the first floppy drive.

2for the second floppy drive.

3for the first hard-disk partition.

4-26for all other drives and partitions.

Remark: Under Linux, and Unix in general, the concept of disk is different than the dos one, since the filesystem is seen as one big directory tree. For this reason, the `DiskFree` and DiskSize (1401) functions must be mimicked using filenames that reside on the partitions. For more information, see AddDisk (1371).

Errors: On error, `-1` is returned.

See also: DiskSize (1401), AddDisk (1371)

Listing: ./sysutex/ex27.pp

```
Program Example27;

{ This program demonstrates the DiskFree function }

Uses sysutils;

Begin
  Write ('Size of current disk       : ',DiskSize(0));
  Writeln (' (= ',DiskSize(0) div 1024,'k)');
  Write ('Free space of current disk : ',Diskfree(0));
  Writeln (' (= ',Diskfree(0) div 1024,'k)');
End.
```

30.12.66 DiskSize

Synopsis: Return the total amount of diskspace.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

Description: `DiskSize` returns the size (in bytes) of disk `Drive`. Drive is the number of the disk drive:

0for the current drive.

1for the first floppy drive.

2for the second floppy drive.

3for the first hard-disk parttion.

4-26for all other drives and partitions.

Remark: Under Linux, and Unix in general, the concept of disk is different than the dos one, since the filesystem is seen as one big directory tree. For this reason, the DiskFree (1401) and `DiskSize` functions must be mimicked using filenames that reside on the partitions. For more information, see AddDisk (1371)

For an example, see DiskFree (1401).

Errors: On error, `-1` is returned.

See also: DiskFree (1401), AddDisk (1371)

30.12.67 DisposeStr

Synopsis: Dispose an ansistring from the heap.

Declaration: `procedure DisposeStr(S: PString)`

Visibility: default

Description: `DisposeStr` removes the dynamically allocated string `S` from the heap, and releases the occupied memory.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

For an example, see DisposeStr (1402).

Errors: None.

See also: NewStr (1449), AppendStr (1385), AssignStr (1386)

30.12.68 DoDirSeparators

Synopsis: Convert known directory separators to the current directory separator.

Declaration: `procedure DoDirSeparators(var FileName: String)`

Visibility: default

Description: This function replaces all known directory separators in `FileName` to the directory separator character for the current system. The list of known separators is specified in the DirSeparators (1350) constant.

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409)

Listing: ./sysutex/ex32.pp

```
Program Example32;

{ This program demonstrates the DoDirSeparators function }
{$H+}

Uses sysutils;

Procedure Testit (F : String);

begin
  WriteLn ('Before : ',F);
  DoDirSeparators (F);
  WriteLn ('After  : ',F);
end;

Begin
  Testit (GetCurrentDir);
  Testit ('c:\pp\bin\win32');
  Testit ('/usr/lib/fpc');
  Testit ('\usr\lib\fpc');
End.
```

30.12.69 EncodeDate

Synopsis: Encode a Year,Month,Day to a `TDateTime` value.

Declaration: `function EncodeDate(Year: Word;Month: Word;Day: Word) : TDateTime`

Visibility: default

Description: `EncodeDate` encodes the `Year`, `Month` and `Day` variables to a date in `TDateTime` format. It does the opposite of the DecodeDate (1398) procedure.

The parameters must lie withing valid ranges (boundaries included):

Yearmust be between 1 and 9999.

Monthmust be within the range 1-12.

Daymsut be between 1 and 31.

Errors: In case one of the parameters is out of it's valid range, an EConvertError (1492) exception is raised.

See also: EncodeTime (1404), DecodeDate (1398)

Listing: ./sysutex/ex11.pp

```
Program Example11;

{ This program demonstrates the EncodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin
  DecodeDate (Date,YY,MM,DD);
  WriteLn ('Today is : ',FormatDateTime ('dd mmmm yyyy',EnCodeDate(YY,Mm,Dd)));
End.
```

30.12.70 EncodeTime

Synopsis: Encode a Hour,Min,Sec,millisec to a `TDateTime` value.

Declaration:
```
function EncodeTime(Hour: Word;Minute: Word;Second: Word;
                    MilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeTime` encodes the `Hour`, `Minute`, `Second`, `MilliSecond` variables to a `TDateTime` format result. It does the opposite of the DecodeTime (1399) procedure.

The parameters must have a valid range (boundaries included):

Hourmust be between 0 and 23.

Minute,secondmust both be between 0 and 59.

Millisecondmust be between 0 and 999.

Errors: In case one of the parameters is out of it's valid range, an EConvertError (1492) exception is raised.

See also: EncodeDate (1403), DecodeTime (1399)

Listing: ./sysutex/ex12.pp

```
Program Example12;

{ This program demonstrates the EncodeTime function }

Uses sysutils;

Var Hh,MM,SS,MS : Word;

Begin
  DeCodeTime (Time,Hh,MM,SS,MS);
  Writeln ('Present Time is : ',FormatDateTime('hh:mm:ss',EnCodeTime (HH,MM,SS,MS)));
End.
```

30.12.71 ExceptAddr

Synopsis: Current exception address.

Declaration:
```
function ExceptAddr : Pointer
```

Visibility: default

Description: `ExceptAddr` returns the address from the currently treated exception object when an exception is raised, and the stack is unwound.

See also: ExceptObject (1405), ExceptionErrorMessage (1405), ShowException (1454)

30.12.72 ExceptFrameCount

Synopsis: Number of frames included in an exception backtrace

Declaration:
```
function ExceptFrameCount : LongInt
```

Visibility: default

Description: `ExceptFrameCount` returns the number of frames that are included in an exception stack frame backtrace. The function returns 0 if there is currently no exception being handled. (i.e. it only makes sense to call this function in an `finally..end` or `except..end` block.

Errors: None.

See also: ExceptFrames (1405), ExceptAddr (1404), ExceptObject (1405), ExceptProc (1350)

30.12.73 ExceptFrames

Synopsis:

Declaration:
```
function ExceptFrames : PPointer
```

Visibility: default

Description:

Errors:

See also: ExceptFrameCount (1404), ExceptAddr (1404), ExceptObject (1405), ExceptProc (1350)

30.12.74 ExceptionErrorMessage

Synopsis: Return a message describing the exception.

Declaration:
```
function ExceptionErrorMessage(ExceptObject: TObject;
                               ExceptAddr: Pointer;Buffer: PChar;
                               Size: Integer) : Integer
```

Visibility: default

Description: `ExceptionErrorMessage` creates a string that describes the exception object `ExceptObject` at address `ExceptAddr`. It can be used to display exception messages. The string will be stored in the memory pointed to by `Buffer`, and will at most have `Size` characters.

The routine checks whether `ExceptObject` is a Exception (1496) object or not, and adapts the output accordingly.

See also: ExceptObject (1405), ExceptAddr (1404), ShowException (1454)

30.12.75 ExceptObject

Synopsis: Current Exception object.

Declaration:
```
function ExceptObject : TObject
```

Visibility: default

Description: `ExceptObject` returns the currently treated exception object when an exception is raised, and the stack is unwound.

Errors: If there is no exception, the function returns `Nil`

See also: ExceptAddr (1404), ExceptionErrorMessage (1405), ShowException (1454)

30.12.76 ExcludeTrailingBackslash

Synopsis: Strip trailing directory separator from a pathname, if needed.

Declaration:
```
function ExcludeTrailingBackslash(const Path: String) : String
```

Visibility: default

Description: `ExcludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use ExcludeTrailingPathDelimiter (1406) instead.

See also: IncludeTrailingPathDelimiter (1442), ExcludeTrailingPathDelimiter (1406), PathDelim (1358), IsPathDelimiter (1445)

30.12.77 ExcludeTrailingPathDelimiter

Synopsis: Strip trailing directory separator from a pathname, if needed.

Declaration:
```
function ExcludeTrailingPathDelimiter(const Path: String) : String
```

Visibility: default

Description: `ExcludeTrailingPathDelimiter` removes the trailing path delimiter character (PathDelim (1358)) from `Path` if it is present, and returns the result.

See also: ExcludeTrailingBackslash (1406), IncludeTrailingPathDelimiter (1442), PathDelim (1358), IsPathDelimiter (1445)

30.12.78 ExecuteProcess

Synopsis: Execute another process (program).

Declaration:
```
function ExecuteProcess(const Path: AnsiString;
                        const ComLine: AnsiString) : Integer
function ExecuteProcess(const Path: AnsiString;
                        const ComLine: Array of AnsiString) : Integer
```

Visibility: default

Description: `ExecuteProcess` will execute the program in `Path`, passing it the arguments in `ComLine`. `ExecuteProcess` will then wait for the program to finish, and will return the exit code of the executed program. In case `ComLine` is a single string, it will be split out in an array of strings, taking into account common whitespace and quote rules.

Errors: In case the program could not be executed or an other error occurs, an EOSError (1494) exception will be raised.

See also: EOSError (1494)

30.12.79 ExpandFileName

Synopsis: Expand a relative filename to an absolute filename.

Declaration:
```
function ExpandFileName(const FileName: String) : String
```

Visibility: default

Description: `ExpandFileName` expands the filename to an absolute filename. It changes all directory separator characters to the one appropriate for the system first.

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408), ExtractRelativePath (1409)

Listing: ./sysutex/ex33.pp

```
Program Example33;

{ This program demonstrates the ExpandFileName function }

Uses sysutils;

Procedure Testit (F : String);

begin
  WriteIn (F,' expands to : ',ExpandFileName(F));
end;

Begin
  Testit('ex33.pp');
  Testit(ParamStr(0));
  Testit('/pp/bin/win32/ppc386');
  Testit('\pp\bin\win32\ppc386');
  Testit('.');
End.
```

30.12.80 ExpandUNCFileName

Synopsis: Expand a relative filename to an absolute UNC filename.

Declaration: `function ExpandUNCFileName(const FileName: String) : String`

Visibility: default

Description: `ExpandUNCFileName` runs ExpandFileName (1406) on `FileName` and then attempts to replace the driveletter by the name of a shared disk.

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408), ExtractRelativePath (1409)

30.12.81 ExtractFileDir

Synopsis: Extract the drive and directory part of a filename.

Declaration: `function ExtractFileDir(const FileName: String) : String`

Visibility: default

Description: `ExtractFileDir` returns only the directory part of `FileName`, including a driveletter. The directory name has NO ending directory separator, in difference with ExtractFilePath (1409).

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408), ExtractRelativePath (1409)

Listing: ./sysutex/ex34.pp

```
Program Example34;

{ This program demonstrates the ExtractFileName function }
{$H+}
Uses sysutils;

Procedure Testit(F : String);

begin
  WriteLn ('FileName       : ',F);
  WriteLn ('Has Name       : ',ExtractFileName(F));
  WriteLn ('Has Path       : ',ExtractFilePath(F));
  WriteLn ('Has Extension  : ',ExtractFileExt(F));
  WriteLn ('Has Directory  : ',ExtractFileDir(F));
  WriteLn ('Has Drive      : ',ExtractFileDrive(F));
end;

Begin
   Testit (Paramstr(0));
   Testit ('/usr/local/bin/mysqld');
   Testit ('c:\pp\bin\win32\ppc386.exe');
   Testit ('/pp/bin/win32/ppc386.exe');
End.
```

30.12.82 ExtractFileDrive

Synopsis: Extract the drive part from a filename.

Declaration: `function ExtractFileDrive(const FileName: String) : String`

Visibility: default

Description: `Extracts` the drive letter from a filename. Note that some operating systems do not support drive letters.

For an example, see ExtractFileDir (1407).

Errors:

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408), ExtractRelativePath (1409)

30.12.83 ExtractFileExt

Synopsis: Return the extension from a filename.

Declaration: `function ExtractFileExt(const FileName: String) : String`

Visibility: default

Description: `ExtractFileExt` returns the extension (including the `.`(dot) character) of `FileName`.

For an example, see ExtractFileDir (1407).

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408), ExtractRelativePath (1409)

30.12.84 ExtractFileName

Synopsis: Extract the filename part from a full path filename.

Declaration:
```
function ExtractFileName(const FileName: String) : String
```

Visibility: default

Description: `ExtractFileName` returns the filename part from `FileName`. The filename consists of all characters after the last directory separator character ('/' or '\') or drive letter.

The full filename can always be reconstucted by concatenating the result of ExtractFilePath (1409) and `ExtractFileName`.

For an example, see ExtractFileDir (1407).

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408), ExtractRelativePath (1409)

30.12.85 ExtractFilePath

Synopsis: Extract the path from a filename.

Declaration:
```
function ExtractFilePath(const FileName: String) : String
```

Visibility: default

Description: `ExtractFilePath` returns the path part (including driveletter) from `FileName`. The path consists of all characters before the last directory separator character ('/' or '\'), including the directory separator itself. In case there is only a drive letter, that will be returned.

The full filename can always be reconstucted by concatenating the result of `ExtractFilePath` and ExtractFileName (1409).

For an example, see ExtractFileDir (1407).

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408), ExtractRelativePath (1409)

30.12.86 ExtractRelativepath

Synopsis: Extract a relative path from a filename, given a base directory.

Declaration:
```
function ExtractRelativepath(const BaseName: String;
                             const DestNAme: String) : String
```

Visibility: default

Description: `ExtractRelativePath` constructs a relative path to go from `BaseName` to `DestName`. If `DestName` is on another drive (Not on Unix-like platforms) then the whole `Destname` is returned. *Note:* This function does not exist in the Delphi unit.

Errors: None.

See also: ExtractFileName (1409), ExtractFilePath (1409), ExtractFileDir (1407), ExtractFileDrive (1408), ExtractFileExt (1408)

Listing: ./sysutex/ex35.pp

```
Program Example35;

{ This program demonstrates the ExtractRelativePath function }

Uses sysutils;

Procedure Testit (FromDir,ToDir : String);

begin
  Write ('From "',FromDir,'" to "',ToDir,'" via "');
  WriteIn (ExtractRelativePath(FromDir,ToDir),'"');
end;

Begin
 Testit ('/pp/src/compiler','/pp/bin/win32/ppc386');
 Testit ('/pp/bin/win32/ppc386','/pp/src/compiler');
 Testit ('e:/pp/bin/win32/ppc386','d:/pp/src/compiler');
 Testit ('e:\pp\bin\win32\ppc386','d:\pp\src\compiler');
End.
```

30.12.87 ExtractShortPathName

Synopsis: Returns a 8.3 path name

Declaration: `function ExtractShortPathName(const FileName: String) : String`

Visibility: default

Description: `ExtractShortPathName` returns a 8.3 compliant filename that represents the same file as `FileName`. On platforms other than windows, this is `FileName` itself.

See also: ExtractFilePath (1409), ExtractFileName (1409)

30.12.88 FileAge

Synopsis: Return the timestamp of a file.

Declaration: `function FileAge(const FileName: String) : LongInt`

Visibility: default

Description: `FileAge` returns the last modification time of file `FileName`. The FileDate format can be transformed to `TDateTime` format with the FileDateToDateTime (1412) function.

Errors: In case of errors, `-1` is returned.

See also: FileDateToDateTime (1412), FileExists (1413), FileGetAttr (1413)

Listing: ./sysutex/ex36.pp

```
Program Example36;

{ This program demonstrates the FileAge function }

Uses sysutils;
```

```
Var S : TDateTime;
    fa : Longint;
Begin
  fa:=FileAge('ex36.pp');
  If Fa<>-1 then
    begin
    S:=FileDateTodateTime(fa);
    WriteIn ('I''m from ',DateTimeToStr(S))
    end;
End.
```

30.12.89 FileClose

Synopsis: Close a file handle.

Declaration: `procedure FileClose(Handle: THandle)`

Visibility: default

Description: `FileClose` closes the file handle `Handle`. After this call, attempting to read or write from the handle will result in an error.

For an example, see FileCreate (1411)

Errors: None.

See also: FileCreate (1411), FileWrite (1418), FileOpen (1415), FileRead (1416), FileTruncate (1418), FileSeek (1417)

30.12.90 FileCreate

Synopsis: Create a new file and return a handle to it.

Declaration: `function FileCreate(const FileName: String) : THandle`
`function FileCreate(const FileName: String;Mode: Integer) : THandle`

Visibility: default

Description: `FileCreate` creates a new file with name `FileName` on the disk and returns a file handle which can be used to read or write from the file with the FileRead (1416) and FileWrite (1418) functions. If a file with name `FileName` already existed on the disk, it is overwritten.

Errors: If an error occurs (e.g. disk full or non-existent path), the function returns `-1`.

See also: FileClose (1411), FileWrite (1418), FileOpen (1415), FileRead (1416), FileTruncate (1418), FileSeek (1417)

Listing: ./sysutex/ex37.pp

```
Program Example37;

{ This program demonstrates the FileCreate function }

Uses sysutils;

Var I,J,F : Longint;

Begin
```

```
  F:=FileCreate ('test.dat');
  If F=-1 then
    Halt(1);
  For I:=0 to 100 do
    FileWrite(F,I,SizeOf(i));
  FileClose(f);
  F:=FileOpen ('test.dat',fmOpenRead);
  For I:=0 to 100 do
    begin
    FileRead (F,J,SizeOF(J));
    If J<>I then
      WriteIn ('Mismatch at file position ',I)
    end;
  FileSeek(F,0,fsFromBeginning);
  Randomize;
  Repeat
    FileSeek(F,Random(100)*4,fsFromBeginning);
    FileRead (F,J,SizeOf(J));
    WriteIn ('Random read : ',j);
  Until J>80;
  FileClose(F);
  F:=FileOpen('test.dat',fmOpenWrite);
  I:=50*SizeOf(Longint);
  If FileTruncate(F,I) then
    WriteIn('SuccessFully truncated file to ',I,' bytes.');
  FileClose(F);
End.
```

30.12.91 FileDateToDateTime

Synopsis: Convert a `FileDate` value to a `TDateTime` value.

Declaration: `function FileDateToDateTime(Filedate: LongInt) : TDateTime`

Visibility: default

Description: `FileDateToDateTime` converts the date/time encoded in `filedate` to a `TDateTime` encoded form. It can be used to convert date/time values returned by the FileAge (1410) or FindFirst (1419)/FindNext (1420) functions to `TDateTime` form.

Errors: None.

See also: DateTimeToFileDate (1394)

Listing: ./sysutex/ex13.pp

```
Program Example13;

{ This program demonstrates the FileDateToDateTime function }

Uses sysutils;

Var
  ThisAge : Longint;

Begin
 Write ('ex13.pp created on :');
 ThisAge:=FileAge('ex13.pp');
```

```
  WriteIn (DateTimeToStr(FileDateToDateTime(ThisAge)));
End.
```

30.12.92 FileExists

Synopsis: Check whether a file exists in the filesystem.

Declaration: `function FileExists(const FileName: String) : Boolean`

Visibility: default

Description: `FileExists` returns `True` if a file with name `FileName` exists on the disk, `False` otherwise.

Errors: None.

See also: FileAge (1410), FileGetAttr (1413), FileSetAttr (1417)

Listing: ./sysutex/ex38.pp

```
Program Example38;

{ This program demonstrates the FileExists function }

Uses sysutils;

Begin
  If FileExists(ParamStr(0)) Then
    WriteIn ('All is well, I seem to exist.');
End.
```

30.12.93 FileGetAttr

Synopsis: Return attributes of a file.

Declaration: `function FileGetAttr(const FileName: String) : LongInt`

Visibility: default

Description: `FileGetAttr` returns the attribute settings of file `FileName`. The attribute is a `OR`-ed combination of the following constants:

faReadOnlyThe file is read-only.

faHiddenThe file is hidden. (On unix, this means that the filename starts with a dot)

faSysFileThe file is a system file (On unix, this means that the file is a character, block or FIFO file).

faVolumeIdVolume Label. Not possible under unix.

faDirectoryFile is a directory.

faArchivefile is an archive. Not possible on Unix

Errors: In case of error, -1 is returned.

See also: FileSetAttr (1417), FileAge (1410), FileGetDate (1414)

Listing: ./sysutex/ex40.pp

```
Program Example40;

{ This program demonstrates the FileGetAttr function }

Uses sysutils;

Procedure Testit (Name : String);

Var F : Longint;

Begin
  F:=FileGetAttr(Name);
  If F<>-1 then
    begin
    Writeln ('Testing : ',Name);
    If (F and faReadOnly)<>0 then
      Writeln ('File is ReadOnly');
    If (F and faHidden)<>0 then
      Writeln ('File is hidden');
    If (F and faSysFile)<>0 then
      Writeln ('File is a system file');
    If (F and faVolumeID)<>0 then
      Writeln ('File is a disk label');
    If (F and faArchive)<>0 then
      Writeln ('File is artchive file');
    If (F and faDirectory)<>0 then
      Writeln ('File is a directory');
    end
  else
   Writeln ('Error reading attribites of ',Name);
end;

begin
  testit ('ex40.pp');
  testit (ParamStr(0));
  testit ('.');
  testit ('/');
End.
```

30.12.94 FileGetDate

Synopsis: Return the file time of an opened file.

Declaration: `function FileGetDate(Handle: THandle) : LongInt`

Visibility: default

Description: `FileGetdate` returns the filetime of the opened file with filehandle `Handle`. It is the same as FileAge (1410), with this difference that `FileAge` only needs the file name, while `FilegetDate` needs an open file handle.

Errors: On error, -1 is returned.

See also: FileAge (1410)

Listing: ./sysutex/ex39.pp

```
Program Example39;

{ This program demonstrates the FileGetDate function }

Uses sysutils;

Var F,D : Longint;

Begin
  F:=FileCreate('test.dat');
  D:=FileGetDate(F);
  Writeln ('File created on ',DateTimeToStr(FileDateToDateTime(D)));
  FileClose(F);
  DeleteFile('test.dat');
End.
```

30.12.95 FileIsReadOnly

Synopsis: Check whether a file is read-only.

Declaration: `function FileIsReadOnly(const FileName: String) : Boolean`

Visibility: default

Description: `FileIsReadOnly` checks whether `FileName` exists in the filesystem and is a read-only file. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: FileExists (1413)

30.12.96 FileOpen

Synopsis: Open an existing file and return a filehandle

Declaration: `function FileOpen(const FileName: String;Mode: Integer) : THandle`

Visibility: default

Description: `FileOpen` opens a file with name `FileName` with mode `Mode`. `Mode` can be one of the following constants:

fmOpenReadThe file is opened for reading.

fmOpenWriteThe file is opened for writing.

fmOpenReadWriteThe file is opened for reading and writing.

If the file has been successfully opened, it can be read from or written to (depending on the `Mode` parameter) with the FileRead (1416) and `FileWrite` functions.

Remark: Remark that you cannot open a file if it doesn't exist yet, i.e. it will not be created for you. If you want tp create a new file, or overwrite an old one, use the FileCreate (1411) function.

For an example, see FileOpen (1415)

Errors: On Error, -1 is returned.

See also: FileClose (1411), FileWrite (1418), FileCreate (1411), FileRead (1416), FileTruncate (1418), FileSeek (1417)

30.12.97 FileRead

Synopsis: Read data from a filehandle in a buffer.

Declaration: `function FileRead(Handle: THandle;var Buffer;Count: LongInt) : LongInt`

Visibility: default

Description: `FileRead` reads `Count` bytes from file-handle `Handle` and stores them into `Buffer`. Buffer must be at least `Count` bytes long. No checking on this is performed, so be careful not to overwrite any memory. `Handle` must be the result of a FileOpen (1415) call.

The function returns the number of bytes actually read, or -1 on error.

For an example, see FileCreate (1411)

Errors: On error, -1 is returned.

See also: FileClose (1411), FileWrite (1418), FileCreate (1411), FileOpen (1415), FileTruncate (1418), FileSeek (1417)

30.12.98 FileSearch

Synopsis: Search for a file in a path.

Declaration: `function FileSearch(const Name: String;const DirList: String) : String`

Visibility: default

Description: `FileSearch` looks for the file `Name` in `DirList`, where dirlist is a list of directories, separated by semicolons or colons. It returns the full filename of the first match found.

Errors: On error, an empty string is returned.

See also: ExpandFileName (1406), FindFirst (1419)

Listing: ./sysutex/ex41.pp

```
Program Example41;

{ Program to demonstrate the FileSearch function. }

Uses Sysutils;

Const
{$ifdef unix}
  FN = 'find';
  P = '.:/bin:/usr/bin';
{$else}
  FN = 'find.exe';
  P = 'c:\dos;c:\windows;c:\windows\system;c:\windows\system32';
{$endif}

begin
  Writeln ('find is in : ',FileSearch (FN,P));
end.
```

30.12.99 FileSeek

Synopsis: Set the current file position on a file handle.

Declaration:
```
function FileSeek(Handle: THandle;FOffset: LongInt;Origin: LongInt)
                  : LongInt
function FileSeek(Handle: THandle;FOffset: Int64;Origin: LongInt)
                  : Int64
```

Visibility: default

Description: `FileSeek` sets the file pointer on position `Offset`, starting from `Origin`. Origin can be one of the following values:

fsFromBeginning`Offset` is relative to the first byte of the file. This position is zero-based. i.e. the first byte is at offset 0.

fsFromCurrent`Offset` is relative to the current position.

fsFromEnd`Offset` is relative to the end of the file. This means that `Offset` can only be zero or negative in this case.

If successfull, the function returns the new file position, relative to the beginning of the file.

Remark: The abovementioned constants do not exist in Delphi.

Errors: On error, -1 is returned.

See also: FileClose (1411), FileWrite (1418), FileCreate (1411), FileOpen (1415), FileRead (1416), FileTruncate (1418)

Listing: ./sysutex/ex42.pp

```
Program Example42;

{ This program demonstrates the FileSetAttr function }

Uses sysutils;

Begin
  If FileSetAttr ('ex40.pp',faReadOnly or faHidden)=0 then
    WriteIn ('Successfully made file hidden and read-only.')
  else
    WriteIn ('Coulnd''t make file hidden and read-only.');
End.
```

30.12.100 FileSetAttr

Synopsis: Set the attributes of a file.

Declaration: `function FileSetAttr(const Filename: String;Attr: LongInt) : LongInt`

Visibility: default

Description: `FileSetAttr` sets the attributes of `FileName` to `Attr`. If the function was successful, 0 is returned, -1 otherwise. `Attr` can be set to an OR-ed combination of the pre-defined `faXXX` constants.

This function is not implemented on Unixes.

Errors: On error, -1 is returned (always on Unixes).

See also: FileGetAttr (1413), FileGetDate (1414), FileSetDate (1418)

30.12.101 FileSetDate

Synopsis: Set the date of a file.

Declaration:
```
function FileSetDate(Handle: THandle;Age: LongInt) : LongInt
function FileSetDate(const FileName: String;Age: LongInt) : LongInt
```

Visibility: default

Description: `FileSetDate` sets the file date of the open file with handle `Handle` or to `Age`, where `Age` is a DOS date-and-time stamp value.

Alternatively, the filename may be specified with the `FileName` argument. This variant of the call is mandatory on unices, since there is no OS support for setting a file timestamp based on a handle. (the handle may not be a real file at all).

The function returns zero if successfull.

Errors: On Unix, the handle variant always returns -1, since this is impossible to implement. On Windows and DOS, a negative error code is returned.

30.12.102 FileTruncate

Synopsis: Truncate an open file to a given size.

Declaration:
```
function FileTruncate(Handle: THandle;Size: Int64) : Boolean
```

Visibility: default

Description: `FileTruncate` truncates the file with handle `Handle` to `Size` bytes. The file must have been opened for writing prior to this call. The function returns `True` is successful, `False` otherwise.

For an example, see FileCreate (1411).

Errors: On error, the function returns `False`.

See also: FileClose (1411), FileWrite (1418), FileCreate (1411), FileOpen (1415), FileRead (1416), FileSeek (1417)

30.12.103 FileWrite

Synopsis: Write data from a buffer to a given filehandle.

Declaration:
```
function FileWrite(Handle: THandle;const Buffer;Count: LongInt)
                         : LongInt
```

Visibility: default

Description: `FileWrite` writes `Count` bytes from `Buffer` to the file with handle `Handle`. Prior to this call, the file must have been opened for writing. `Buffer` must be at least `Count` bytes large, or a memory access error may occur.

The function returns the number of bytes written, or -1 in case of an error.

For an example, see FileCreate (1411).

Errors: In case of error, -1 is returned.

See also: FileClose (1411), FileCreate (1411), FileOpen (1415), FileRead (1416), FileTruncate (1418), FileSeek (1417)

30.12.104 FindClose

Synopsis: Close a find handle

Declaration:
```
procedure FindClose(var F: TSearchRec)
```

Visibility: default

Description: `FindClose` ends a series of FindFirst (1419)/FindNext (1420) calls, and frees any memory used by these calls. It is *absolutely* necessary to do this call, or huge memory losses may occur.

For an example, see FindFirst (1419).

Errors: None.

See also: FindFirst (1419), FindNext (1420)

30.12.105 FindCmdLineSwitch

Synopsis: Check whether a certain switch is present on the command-line.

Declaration:
```
function FindCmdLineSwitch(const Switch: String;
                           const Chars: TSysCharSet;IgnoreCase: Boolean)
                            : Boolean
function FindCmdLineSwitch(const Switch: String;IgnoreCase: Boolean)
                            : Boolean
function FindCmdLineSwitch(const Switch: String) : Boolean
```

Visibility: default

Description: `FindCmdLineSwitch` will check all command-line arguments for the presence of the option `Switch`. It will return `True` if it was found, `False` otherwise. Characters that appear in `Chars` (default is SwitchChars (1360)) are assumed to indicate an option (switch). If the parameter `IgnoreCase` is `True`, case will be ignored when looking for the switch. Default is to search case sensitive.

Errors: None.

See also: SwitchChars (1360)

30.12.106 FindFirst

Synopsis: Start a file search and return a findhandle

Declaration:
```
function FindFirst(const Path: String;Attr: LongInt;
                   out Rslt: TSearchRec) : LongInt
```

Visibility: default

Description: `FindFirst` looks for files that match the name (possibly with wildcards) in `Path` and extra attributes `Attr`. It then fills up the `Rslt` record with data gathered about the file. It returns 0 if a file matching the specified criteria is found, a nonzero value (-1 on Unix-like platforms) otherwise.

`Attr` is an or-ed combination of the following constants:

faReadOnlyThe file is read-only.

faHiddenThe file is hidden. (On unix, this means that the filename starts with a dot)

faSysFileThe file is a system file (On unix, this means that the file is a character, block or FIFO file).

faVolumeIdDrive volume Label. Not possible under unix.

faDirectoryFile is a directory.

faArchivefile is an archive. Not possible on Unix

It is a common misconception that `Attr` specifies a set of attributes which must be matched in order for a file to be included in the list. This is not so: The value of `Attr` specifies *additional* attributes, this means that the returned files are either normal files or have an attribute which is present in `Attr`.

Specifically: specifying `faDirectory` as a value for `Attr` does not mean that only directories will be returned. Normal files *and* directories will be returned.

The `Rslt` record can be fed to subsequent calls to `FindNext`, in order to find other files matching the specifications.

Remark: A `FindFirst` call must *always* be followed by a FindClose (1419) call with the same `Rslt` record. Failure to do so will result in memory loss.

Errors: On error the function returns -1 on Unix-like platforms, a nonzero error code on Windows.

See also: FindClose (1419), FindNext (1420)

Listing: ./sysutex/ex43.pp

```
Program Example43;

{ This program demonstrates the FindFirst function }

Uses SysUtils;

Var Info : TSearchRec;
    Count : Longint;

Begin
  Count:=0;
  If FindFirst ('*',faAnyFile and faDirectory,Info)=0 then
    begin
    Repeat
      Inc(Count);
      With Info do
        begin
        If (Attr and faDirectory) = faDirectory then
          Write('Dir : ');
        WriteIn (Name:40,Size:15);
        end;
    Until FindNext(info)<>0;
    end;
  FindClose(Info);
  WriteIn ('Finished search. Found ',Count,' matches');

End.
```

30.12.107 FindNext

Synopsis: Find the next entry in a findhandle.

Declaration: `function FindNext(var Rslt: TSearchRec) : LongInt`

Visibility: default

Description: `FindNext` finds a next occurrence of a search sequence initiated by `FindFirst`. If another record matching the criteria in Rslt is found, 0 is returned, a nonzero constant is returned otherwise.

Remark: The last `FindNext` call must *always* be followed by a `FindClose` call with the same `Rslt` record. Failure to do so will result in memory loss.

For an example, see FindFirst (1419)

Errors: On error (no more file is found), a nonzero constant is returned.

See also: FindFirst (1419), FindClose (1419)

30.12.108 FloattoCurr

Synopsis: Convert a float to a Currency value.

Declaration:
```
function FloattoCurr(const Value: Extended) : Currency
```

Visibility: default

Description: `FloatToCurr` converts the `Value` floating point value to a `Currency` value. It checks whether `Value` is in the valid range of currencies (determined by MinCurrency (1358) and MaxCurrency (1357)). If not, an EConvertError (1492) exception is raised.

Errors: If `Value` is out of range, an EConvertError (1492) exception is raised.

See also: EConvertError (1492), TryFloatToCurr (1484), MinCurrency (1358), MaxCurrency (1357)

30.12.109 FloatToDateTime

Synopsis: Convert a float to a `TDateTime` value.

Declaration:
```
function FloatToDateTime(const Value: Extended) : TDateTime
```

Visibility: default

Description: `FloatToDateTime` converts the `Value` floating point value to a `TDateTime` value. It checks whether `Value` is in the valid range of dates (determined by MinDateTime (1358) and MaxDateTime (1358)). If not, an EConvertError (1492) exception is raised.

Errors: If `Value` is out of range, an EConvertError (1492) exception is raised.

See also: EConvertError (1492), MinDateTime (1358), MaxDateTime (1358)

30.12.110 FloatToDecimal

Synopsis: Convert a float value to a `TFloatRec` value.

Declaration:
```
procedure FloatToDecimal(out Result: TFloatRec;const Value;
                         ValueType: TFloatValue;Precision: Integer;
                         Decimals: Integer)
procedure FloatToDecimal(out Result: TFloatRec;Value: Extended;
                         Precision: Integer;Decimals: Integer)
```

Visibility: default

Description: `FloatToDecimal` converts the float `Value` to a float description in the `Result`TFloatRec (1364) format. It will store `Precision` digits in the `Digits` field, of which at most `Decimal` decimals.

Errors: None.

See also: TFloatRec (1364)

30.12.111 FloatToStr

Synopsis: Convert a float value to a string using a fixed format.

Declaration:
```
function FloatToStr(Value: Double) : String
function FloatToStr(Value: Double;const FormatSettings: TFormatSettings)
                    : String
function FloatToStr(Value: Single) : String
function FloatToStr(Value: Single;const FormatSettings: TFormatSettings)
                    : String
function FloatToStr(Value: Currency) : String
function FloatToStr(Value: Currency;
                  const FormatSettings: TFormatSettings) : String
function FloatToStr(Value: Comp) : String
function FloatToStr(Value: Comp;const FormatSettings: TFormatSettings)
                    : String
function FloatToStr(Value: Int64) : String
function FloatToStr(Value: Int64;const FormatSettings: TFormatSettings)
                    : String
```

Visibility: default

Description: `FloatToStr` converts the floating point variable `Value` to a string representation. It will choose the shortest possible notation of the two following formats:

Fixed formatwill represent the string in fixed notation,

Decimal formatwill represent the string in scientific notation.

More information on these formats can be found in FloatToStrF (1423). `FloatToStr` is completely equivalent to the following call:

```
FloatToStrF(Value, ffGeneral,15, 0);
```

Errors: None.

See also: FloatToStrF (1423), FormatFloat (1434), StrToFloat (1472)

Listing: ./sysutex/ex67.pp

```
Program Example67;

{ This program demonstrates the FloatToStr function }

Uses sysutils;

Procedure Testit (Value : Extended);

begin
  WriteIn (Value,' -> ',FloatToStr(Value));
  WriteIn (-Value,' -> ',FloatToStr(-Value));
end;
```

```
Begin
  Testit (0.0);
  Testit (1.1);
  Testit (1.1e-3);
  Testit (1.1e-20);
  Testit (1.1e-200);
  Testit (1.1e+3);
  Testit (1.1e+20);
  Testit (1.1e+200);
End.
```

30.12.112 FloatToStrF

Synopsis: Convert a float value to a string using a given format.

Declaration:
```
function FloatToStrF(Value: Double;format: TFloatFormat;
                     Precision: Integer;Digits: Integer) : String
function FloatToStrF(Value: Double;format: TFloatFormat;
                     Precision: Integer;Digits: Integer;
                     const FormatSettings: TFormatSettings) : String
function FloatToStrF(Value: Single;format: TFloatFormat;
                     Precision: Integer;Digits: Integer) : String
function FloatToStrF(Value: Single;format: TFloatFormat;
                     Precision: Integer;Digits: Integer;
                     const FormatSettings: TFormatSettings) : String
function FloatToStrF(Value: Comp;format: TFloatFormat;
                     Precision: Integer;Digits: Integer) : String
function FloatToStrF(Value: Comp;format: TFloatFormat;
                     Precision: Integer;Digits: Integer;
                     const FormatSettings: TFormatSettings) : String
function FloatToStrF(Value: Currency;format: TFloatFormat;
                     Precision: Integer;Digits: Integer) : String
function FloatToStrF(Value: Currency;format: TFloatFormat;
                     Precision: Integer;Digits: Integer;
                     const FormatSettings: TFormatSettings) : String
function FloatToStrF(Value: Int64;format: TFloatFormat;
                     Precision: Integer;Digits: Integer) : String
function FloatToStrF(Value: Int64;format: TFloatFormat;
                     Precision: Integer;Digits: Integer;
                     const FormatSettings: TFormatSettings) : String
```

Visibility: default

Description: `FloatToStrF` converts the floating point number `value` to a string representation, according to the settings of the parameters `Format`, `Precision` and `Digits`.

The meaning of the `Precision` and `Digits` parameter depends on the `Format` parameter. The format is controlled mainly by the `Format` parameter. It can have one of the following values:

ffcurrency Money format. `Value` is converted to a string using the global variables `CurrencyString`, `CurrencyFormat` and `NegCurrencyFormat`. The `Digits` paramater specifies the number of digits following the decimal point and should be in the range -1 to 18. If Digits equals `-1`, `CurrencyDecimals` is assumed. The `Precision` parameter is ignored.

ffExponent Scientific format. `Value` is converted to a string using scientific notation: 1 digit before the decimal point, possibly preceded by a minus sign if `Value` is negative. The number of digits after the decimal point is controlled by `Precision` and must lie in the range 0 to 15.

ffFixedFixed point format. `Value` is converted to a string using fixed point notation. The result is composed of all digits of the integer part of `Value`, preceded by a minus sign if `Value` is negative. Following the integer part is `DecimalSeparator` and then the fractional part of `Value`, rounded off to `Digits` numbers. If the number is too large then the result will be in scientific notation.

ffGeneralGeneral number format. The argument is converted to a string using `ffExponent` or `ffFixed` format, depending on wich one gives the shortest string. There will be no trailing zeroes. If `Value` is less than `0.00001` or if the number of decimals left of the decimal point is larger than `Precision` then scientific notation is used, and `Digits` is the minimum number of digits in the exponent. Otherwise `Digits` is ignored.

ffnumberIs the same as `ffFixed`, except that thousand separators are inserted in the resultig string.

Errors: None.

See also: FloatToStr (1422), FloatToText (1425)

Listing: ./sysutex/ex68.pp

```
Program Example68;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : Array [TFloatFormat] of string[10] =
          ('general','exponent','fixed','number','Currency');

Procedure Testit (Value :  Extended);

Var I,J : longint;
    FF : TFloatFormat;

begin
  For I:=5 to 15 do
    For J:=1 to 4 do
      For FF:=ffgeneral to ffcurrency do
        begin
        Write (Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        WriteIn (FloatToStrf(Value,FF,I,J));
        Write (-Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        WriteIn (FloatToStrf(-Value,FF,I,J));
        end;
end;

Begin
  Testit (1.1);
  Testit (1.1E1);
  Testit (1.1E-1);
  Testit (1.1E5);
  Testit (1.1E-5);
  Testit (1.1E10);
  Testit (1.1E-10);
  Testit (1.1E15);
  Testit (1.1E-15);
  Testit (1.1E100);
  Testit (1.1E-100);
End.
```

30.12.113 FloatToText

Synopsis: Return a string representation of a float, with a given format.

Declaration:
```
function FloatToText(Buffer: PChar;Value: Extended;format: TFloatFormat;
                     Precision: Integer;Digits: Integer) : LongInt
function FloatToText(Buffer: PChar;Value: Extended;format: TFloatFormat;
                     Precision: Integer;Digits: Integer;
                     const FormatSettings: TFormatSettings) : LongInt
```

Visibility: default

Description: `FloatToText` converts the floating point variable `Value` to a string representation and stores it in `Buffer`. The conversion is governed by `format`, `Precisison` and `Digits`. more information on these parameters can be found in FloatToStrF (1423). `Buffer` should point to enough space to hold the result. No checking on this is performed.

The result is the number of characters that was copied in `Buffer`.

Errors: None.

See also: FloatToStr (1422), FloatToStrF (1423)

Listing: ./sysutex/ex69.pp

```
Program Example68;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : Array [TFloatFormat] of string[10] =
        ('general','exponent','fixed','number','Currency');

Procedure Testit (Value :  Extended);

Var I,J : longint;
    FF : TFloatFormat;
    S : ShortString;

begin
  For I:=5 to 15 do
    For J:=1 to 4 do
      For FF:=ffgeneral to ffcurrency do
        begin
        Write (Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        SetLength(S,FloatToText (@S[1],Value,FF,I,J));
        WriteIn (S);
        Write (-Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        SetLength(S,FloatToText (@S[1],-Value,FF,I,J));
        WriteIn (S);
        end;
end;

Begin
  Testit (1.1);
  Testit (1.1E1);
  Testit (1.1E-1);
  Testit (1.1E5);
  Testit (1.1E-5);
```

```
  Testit (1.1E10);
  Testit (1.1E-10);
  Testit (1.1E15);
  Testit (1.1E-15);
  Testit (1.1E100);
  Testit (1.1E-100);
End.
```

30.12.114 FloatToTextFmt

Synopsis: Convert a float value to a string using a given mask.

Declaration:
```
function FloatToTextFmt(Buffer: PChar;Value: Extended;format: PChar;
                        FormatSettings: TFormatSettings) : Integer
function FloatToTextFmt(Buffer: PChar;Value: Extended;format: PChar)
                        : Integer
```

Visibility: default

Description: `FloatToTextFmt` returns a textual representation of `Value` in the memory location pointed to by `Buffer`. it uses the formatting specification in `Format` to do this. The return value is the number of characters that were written in the buffer.

For a list of valid formatting characters, see FormatFloat (1434)

Errors: No length checking is performed on the buffer. The buffer should point to enough memory to hold the complete string. If this is not the case, an access violation may occur.

See also: FormatFloat (1434)

30.12.115 FmtStr

Synopsis: Format a string with given arguments.

Declaration:
```
procedure FmtStr(var Res: String;const Fmt: String;
                 const args: Array of const)
procedure FmtStr(var Res: String;const Fmt: String;
                 const args: Array of const;
                 const FormatSettings: TFormatSettings)
```

Visibility: default

Description: `FmtStr` calls Format (1427) with `Fmt` and `Args` as arguments, and stores the result in `Res`. For more information on how the resulting string is composed, see Format (1427).

Errors: In case of error, a `EConvertError` exception is raised.

See also: Format (1427), FormatBuf (1432)

Listing: ./sysutex/ex70.pp

```
Program Example70;

{ This program demonstrates the FmtStr function }

Uses sysutils;
```

```
Var S : AnsiString;

Begin
  S:='';
  FmtStr (S,'For some nice examples of fomatting see %s.',['Format']);
  Writeln (S);
End.
```

30.12.116 ForceDirectories

Synopsis: Create a chain of directories

Declaration: `function ForceDirectories(const Dir: String) : Boolean`

Visibility: default

Description: `ForceDirectories` tries to create any missing directories in `Dir` till the whole path in `Dir` exists. It returns `True` if `Dir` already existed or was created succesfully. If it failed to create any of the parts, `False` is returned.

30.12.117 Format

Synopsis: Format a string with given arguments.

Declaration:
```
function Format(const Fmt: String;const Args: Array of const) : String
function Format(const Fmt: String;const Args: Array of const;
                const FormatSettings: TFormatSettings) : String
```

Visibility: default

Description: Format replaces all placeholders in`Fmt` with the arguments passed in `Args` and returns the resulting string. A placeholder looks as follows:

```
'%' [[Index]':'] ['-'] [Width] ['.' Precision] ArgType
```

elements between single quotes must be typed as shown without the quotes, and elements between square brackets `[ ]` are optional. The meaning of the different elements is shown below:

'%'starts the placeholder. If you want to insert a literal `%` character, then you must insert two of them : `%%`.

Index ':'takes the `Index`-th element in the argument array as the element to insert. If `index` is omitted, then the zeroth argument is taken.

'-'tells `Format` to left-align the inserted text. The default behaviour is to right-align inserted text. This can only take effect if the `Width` element is also specified.

Widththe inserted string must have at least have `Width` characters. If not, the inserted string will be padded with spaces. By default, the string is left-padded, resulting in a right-aligned string. This behaviour can be changed by the `'-'` character.

'.' Precision Indicates the precision to be used when converting the argument. The exact meaning of this parameter depends on `ArgType`.

The `Index`, `Width` and `Precision` parameters can be replaced by `*`, in which case their value will be read from the next element in the `Args` array. This value must be an integer, or an `EConvertError` exception will be raised.

The argument type is determined from `ArgType`. It can have one of the following values (case insensitive):

DDecimal format. The next argument in the `Args` array should be an integer. The argument is converted to a decimal string. If precision is specified, then the string will have at least `Precision` digits in it. If needed, the string is (left) padded with zeroes.

EScientific format. The next argument in the `Args` array should be a Floating point value. The argument is converted to a decimal string using scientific notation, using FloatToStrF (1423), where the optional precision is used to specify the total number of decimals. (defalt a valueof 15 is used). The exponent is formatted using maximally 3 digits.

In short, the `E` specifier formats it's argument as follows:

```
FloatToStrF(Argument,ffexponent,Precision,3)
```

FFixed point format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string, using fixed notation (see FloatToStrF (1423)). `Precision` indicates the number of digits following the decimal point.

In short, the `F` specifier formats it's argument as follows:

```
FloatToStrF(Argument,ffFixed,ffixed,9999,Precision)
```

GGeneral number format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string using fixed point notation or scientific notation, depending on which gives the shortest result. `Precision` is used to determine the number of digits after the decimal point.

In short, the `G` specifier formats it's argument as follows:

```
FloatToStrF(Argument,ffGeneral,Precision,3)
```

MCurrency format. the next argument in the var{Args} array must be a floating point value. The argument is converted to a decimal string using currency notation. This means that fixed-point notation is used, but that the currency symbol is appended. If precision is specified, then then it overrides the `CurrencyDecimals` global variable used in the FloatToStrF (1423)

In short, the `M` specifier formats it's argument as follows:

```
FloatToStrF(Argument,ffCurrency,9999,Precision)
```

NNumber format. This is the same as fixed point format, except that thousand separators are inserted in the resulting string.

PPointer format. The next argument in the `Args` array must be a pointer (typed or untyped). The pointer value is converted to a string of length 8, representing the hexadecimal value of the pointer.

SString format. The next argument in the `Args` array must be a string. The argument is simply copied to the result string. If `Precision` is specified, then only `Precision` characters are copied to the result string.

UUnsigned decimal format. The next argument in the `Args` array should be an unsigned integer. The argument is converted to a decimal string. If precision is specified, then the string will have at least `Precision` digits in it. If needed, the string is (left) padded with zeroes.

Xhexadecimal format. The next argument in the `Args` array must be an integer. The argument is converted to a hexadecimal string with just enough characters to contain the value of the integer. If `Precision` is specified then the resulting hexadecimal representation will have at least `Precision` characters in it (with a maximum value of 32).

Errors: In case of error, an `EConversionError` exception is raised. Possible errors are:

1. Errors in the format specifiers.
2. The next argument is not of the type needed by a specifier.

3.The number of arguments is not sufficient for all format specifiers.

See also: FormatBuf (1432)

Listing: ./sysutex/ex71.pp

```
Program example71;

{$mode objfpc}

{ This program demonstrates the Format function }

Uses sysutils;

Var P : Pointer;
    fmt,S : string;

Procedure TestInteger;

begin
  Try
    Fmt:='[%d]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%%]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10d]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    fmt:='[%.4d]';S:=Format (fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4d]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:d]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10d]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4d]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10d]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4d]';S:=Format (fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*d]';S:=Format (fmt,[4,5,10]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteIn ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

Procedure TestHexaDecimal;

begin
  try
    Fmt:='[%x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4x]';S:=Format (fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*x]';S:=Format (fmt,[4,5,10]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteIn ('Exception caught : ',E.Message);
```

```
      end;
  end;
  writeln ('Press enter');
  readln;
end;

Procedure TestPointer;

begin
  P:=Pointer(1234567);
  try
    Fmt:='[0x%p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:-10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:-10.4p]';S:=Format (fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*p]';S:=Format (fmt,[4,5,P]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      Writeln ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

Procedure TestString;

begin
  try
    Fmt:='[%s]';S:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:-18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:18.12s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%-*.*s]';s:=Format(fmt,[18,12,'This is a string']);Writeln(fmt:12,'=> ',s);
  except
    On E : Exception do
      begin
      Writeln ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

Procedure TestExponential;

begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
```

```
    Fmt:='[%0:10e]';S:=Format(Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format(Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format(Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format(fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format(fmt,[4,5,1.234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteLn('Exception caught : ',E.Message);
      end;
  end;
  writeln('Press enter');
  readln;
end;

Procedure TestNegativeExponential;

begin
  Try
    Fmt:='[%e]';S:=Format(Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format(Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format(Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format(Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format(Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format(Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format(Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format(fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format(fmt,[4,5,-1.234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteLn('Exception caught : ',E.Message);
      end;
  end;
  writeln('Press enter');
  readln;
end;

Procedure TestSmallExponential;

begin
  Try
    Fmt:='[%e]';S:=Format(Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format(Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format(Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format(Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format(Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format(Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format(Fmt,[0.0123]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format(fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format(fmt,[4,5,0.01234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteLn('Exception caught : ',E.Message);
      end;
  end;
  writeln('Press enter');
```

```
  readln;
end;

Procedure TestSmallNegExponential;

begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format (fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format (fmt,[4,5,-0.01234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteLn ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

begin
  TestInteger;
  TestHexadecimal;
  TestPointer;
  TestExponential;
  TestNegativeExponential;
  TestSmallExponential;
  TestSmallNegExponential;
  teststring;
end.
```

30.12.118 FormatBuf

Synopsis: Format a string with given arguments and store the result in a buffer.

Declaration:
```
function FormatBuf(var Buffer;BufLen: Cardinal;const Fmt;
                   fmtLen: Cardinal;const Args: Array of const)
                    : Cardinal
function FormatBuf(var Buffer;BufLen: Cardinal;const Fmt;
                   fmtLen: Cardinal;const Args: Array of const;
                   const FormatSettings: TFormatSettings) : Cardinal
```

Visibility: default

Description: `FormatBuf` calls Format (1427) and stores the result in `Buf`.

See also: Format (1427)

Listing: ./sysutex/ex72.pp

```
Program Example72;
```

```
{ This program demonstrates the FormatBuf function }

Uses sysutils;

Var
  S : ShortString;

Const
  Fmt : ShortString =  'For some nice examples of fomatting see %s.';

Begin
  S:='';
  SetLength(S,FormatBuf (S[1],255,Fmt[1],Length(Fmt),['Format']));
  Writeln (S);
End.
```

30.12.119 FormatCurr

Synopsis: Format a currency

Declaration:
```
function FormatCurr(const Format: String;Value: Currency) : String
function FormatCurr(const Format: String;Value: Currency;
                    const FormatSettings: TFormatSettings) : String
```

Visibility: default

Description: `FormatCurr` formats the currency `Value` according to the formatting rule in `Format`, and returns the resulting string.

For an explanation of the formatting characters, see FormatFloat (1434).

See also: FormatFloat (1434), FloatToText (1425)

30.12.120 FormatDateTime

Synopsis: Return a string representation of a `TDateTime` value with a given format.

Declaration:
```
function FormatDateTime(FormatStr: String;DateTime: TDateTime) : String
```

Visibility: default

Description: `FormatDateTime` formats the date and time encoded in `DateTime` according to the formatting given in `FormatStr`. The complete list of formatting characters can be found in formatchars (1353).

Errors: On error (such as an invalid character in the formatting string), and `EConvertError` exception is raised.

See also: DateTimeToStr (1395), DateToStr (1397), TimeToStr (1481), StrToDateTime (1471)

Listing: ./sysutex/ex14.pp

```
Program Example14;

{ This program demonstrates the FormatDateTime function }

Uses sysutils;
```

```
Var ThisMoment : TDateTime;

Begin
  ThisMoment:=Now;
  Writeln ('Now : ',FormatDateTime('hh:nn',ThisMoment));
  Writeln ('Now : ',FormatDateTime('DD MM YYYY',ThisMoment));
  Writeln ('Now : ',FormatDateTime('c',ThisMoment));
End.
```

30.12.121 FormatFloat

Synopsis: Format a float according to a certain mask.

Declaration:
```
function FormatFloat(const Format: String;Value: Extended) : String
function FormatFloat(const Format: String;Value: Extended;
                     const FormatSettings: TFormatSettings) : String
```

Visibility: default

Description: `FormatFloat` formats the floating-point value given by `Value` using the format specifications in `Format`. The format specifier can give format specifications for positive, negative or zero values (separated by a semicolon).

If the formatspecifier is empty or the value needs more than 18 digits to be correctly represented, the result is formatted with a call to FloatToStrF (1423) with the `ffGeneral` format option.

The following format specifiers are supported:

0 is a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the 0. If not, the 0 is left as-is.

is also a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the #. If not, it is removed. by a space.

. determines the location of the decimal point. Only the first '.' character is taken into account. If the value contains digits after the decimal point, then it is replaced by the value of the `DecimalSeparator` character.

, determines the use of the thousand separator character in the output string. If the format string contains one or more ',' charactes, then thousand separators will be used. The `ThousandSeparator` character is used.

E+ determines the use of scientific notation. If 'E+' or 'E-' (or their lowercase counterparts) are present then scientific notation is used. The number of digits in the output string is determined by the number of `0` characters after the '`E+`'

; This character separates sections for positive, negative, and zero numbers in the format string.

Errors: If an error occurs, an exception is raised.

See also: FloatToStr (1422)

Listing: ./sysutex/ex89.pp

```
Program Example89;

{ This program demonstrates the FormatFloat function }

Uses sysutils;
```

```
Const
  NrFormat=9;
  FormatStrings : Array[1..NrFormat] of string = (
        '',
        '0',
        '0.00',
        '#.##',
        '#,##0.00',
        '#,##0.00;(#,##0.00)',
        '#,##0.00;;Zero',
        '0.000E+00',
        '#.###E-0');
  NrValue = 5;
  FormatValues : Array[1..NrValue] of Double =
    (1234,-1234,0.5,0,-0.5);

  Width  = 12;
  FWidth = 20;

Var
  I,J : Integer;
  S : String;

begin
  Write('Format':FWidth);
  For I:=1 to NrValue do
    Write(FormatValues[i]:Width:2);
  Writeln;
  For I:=1 to NrFormat do
    begin
    Write(FormatStrings[i]:FWidth);
    For J:=1 to NrValue do
      begin
      S:=FormatFloat(FormatStrings[I],FormatValues[j]);
      Write(S:Width);
      end;
    Writeln;
    end;
End.
```

30.12.122 FreeAndNil

Synopsis: Free object if needed, and set object reference to `Nil`

Declaration: `procedure FreeAndNil(var obj)`

Visibility: default

Description: `FreeAndNil` will free the object in `Obj` and will set the reference in `Obj` to `Nil`. The reference is set to `Nil` first, so if an exception occurs in the destructor of the object, the reference will be `Nil` anyway.

Errors: Exceptions that occur during the destruction of `Obj` are not caught.

30.12.123 GetAppConfigDir

Synopsis: Return the appropriate directory for the application's configuration files.

Declaration:
```
function GetAppConfigDir(Global: Boolean) : String
```

Visibility: default

Description: `GetAppConfigDir` returns the name of a directory in which the application should store its configuration files on the current OS. If the parameter `Global` is `True` then the directory returned is a global directory, i.e. valid for all users on the system. If the parameter `Global` is false, then the directory is specific for the user who is executing the program. On systems that do not support multi-user environments, these two directories may be the same.

The directory which is returned is the name of the directory where the application is supposed to store files. This does not mean that the directory exists, or that the user can write in this directory (especially if `Global=True`). It just returns the name of the appropriate location.

On systems where the operating system provides a call to determine this location, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the ApplicationName (1386) call, but can be configured by means of the OnGetApplicationName (1369) callback.

Errors: None.

See also: GetAppConfigFile (1436), ApplicationName (1386), OnGetApplicationName (1369), CreateDir (1392), SysConfigDir (1360)

30.12.124 GetAppConfigFile

Synopsis: Return an appropriate name for an application configuration file.

Declaration:
```
function GetAppConfigFile(Global: Boolean) : String
function GetAppConfigFile(Global: Boolean;SubDir: Boolean) : String
```

Visibility: default

Description: `GetAppConfigFile` returns the name of a file in which the application can store its configuration parameters. The `Global` parameter determines whether it is a global configuration file (value `True`) or a personal configuration file (value `False`). The parameter `SubDir`, in case it is set to `True`, will insert the name of a directory before the filename. This can be used in case the application needs to store other data than configuration data in an application-specific directory. Default behaviour is to set this to `False`.

No assumptions should be made about the existence or writeability of this file, or the directory where the file should reside.

On systems where the operating system provides a call to determine the location of configuration files, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the ApplicationName (1386) call, but can be configured by means of the OnGetApplicationName (1369) callback.

Errors: None.

See also: GetAppConfigDir (1436), OnGetApplicationName (1369), ApplicationName (1386), CreateDir (1392), ConfigExtension (1355), SysConfigDir (1360)

30.12.125 GetCurrentDir

Synopsis: Return the current working directory of the application.

Declaration: `function GetCurrentDir : String`

Visibility: default

Description: `GetCurrentDir` returns the current working directory.

Errors: None.

See also: SetCurrentDir (1453), DiskFree (1401), DiskSize (1401)

Listing: ./sysutex/ex28.pp

```
Program Example28;

{ This program demonstrates the GetCurrentDir function }

Uses sysutils;

Begin
  WriteIn ('Current Directory is : ',GetCurrentDir);
End.
```

30.12.126 GetDirs

Synopsis: Return a list of directory names from a path.

Declaration: `function GetDirs(var DirName: String;var Dirs: Array of pchar) : LongInt`

Visibility: default

Description: `GetDirs` splits DirName in a null-byte separated list of directory names, `Dirs` is an array of `PChars`, pointing to these directory names. The function returns the number of directories found, or -1 if none were found. DirName must contain only OSDirSeparator as Directory separator chars.

Errors: None.

See also: ExtractRelativePath (1409)

Listing: ./sysutex/ex45.pp

```
Program Example45;

{ This program demonstrates the GetDirs function }
{$H+}

Uses sysutils;

Var Dirs : Array[0..127] of pchar;
    I,Count : longint;
    Dir,NewDir : String;

Begin
  Dir:=GetCurrentDir;
  WriteIn ('Dir : ',Dir);
  NewDir:='';
```

```
  count:=GetDirs(Dir,Dirs);
  For I:=0 to Count-1 do
    begin
    NewDir:=NewDir+'/'+StrPas(Dirs[I]);
    WriteIn (NewDir);
    end;
End.
```

30.12.127 GetEnvironmentString

Synopsis: Return an environment variable by index.

Declaration: `function GetEnvironmentString(Index: Integer) : String`

Visibility: default

Description: `GetEnvironmentString` returns the `Index`-th environment variable. The index is 1 based, and is bounded from above by the result of GetEnvironmentVariableCount (1438).

For an example, GetEnvironmentVariableCount (1438).

Remark: Note that on Windows, environment strings can start with an equal sign (=). This is a trick used to pass the current working directory to a newly created proces. In this case, extracting the variable name as the characters before the first equal sign will result in an empty name.

Errors: If there is no environment, an empty string is returned.

See also: GetEnvironmentVariable (1438), GetEnvironmentVariableCount (1438)

30.12.128 GetEnvironmentVariable

Synopsis: Return the value of an environment variable.

Declaration: `function GetEnvironmentVariable(const EnvVar: String) : String`

Visibility: default

Description: `GetEnvironmentVariable` returns the value of the `EnvVar` environment variable. If the specified variable does not exist or `EnvVar` is empty, an empty string is returned.

See also: GetEnvironmentString (1438), GetEnvironmentVariableCount (1438)

30.12.129 GetEnvironmentVariableCount

Synopsis: Return the number of variables in the environment.

Declaration: `function GetEnvironmentVariableCount : Integer`

Visibility: default

Description: `GetEnvironmentVariableCount` returns the number of variables in the environment. The number is 1 based, but the result may be zero if there are no environment variables.

Errors: If there is no environment, -1 may be returned.

See also: GetEnvironmentString (1438), GetEnvironmentVariable (1438)

Listing: ./sysutex/ex92.pp

```
{$h+}
program example92;

{ This program demonstrates the
  GetEnvironmentVariableCount function }

uses sysutils;

Var
  I : Integer;

begin
  For I:=1 to GetEnvironmentVariableCount do
    WriteIn(i:3,' : ',GetEnvironmentString(i));
end.
```

30.12.130 GetFileHandle

Synopsis: Extract OS handle from an untyped file or text file.

Declaration:
```
function GetFileHandle(var f: File) : LongInt
function GetFileHandle(var f: Text) : LongInt
```

Visibility: default

Description: `GetFileHandle` returns the operating system handle for the file descriptor `F`. It can be used in various file operations which are not directly supported by the pascal language.

30.12.131 GetLastOSError

Synopsis: Return the last code from the OS.

Declaration:
```
function GetLastOSError : Integer
```

Visibility: default

Description: `GetLastOSError` returns the error code from the last operating system call. It does not reset this code. In general, it should be called when an operating system call reported an error condition. In that case, `GetLastOSError` gives extended information about the error.

No assumptions should be made about the resetting of the error code by subsequent OS calls. This may be platform dependent.

See also: RaiseLastOSError (1450)

30.12.132 GetLocalTime

Synopsis: Get the local time.

Declaration:
```
procedure GetLocalTime(var SystemTime: TSystemTime)
```

Visibility: default

Description: `GetLocalTime` returns the system time in a TSystemTime (1350) format.

Errors: None.

See also: Now (1449), Date (1394), Time (1479), TSystemTime (1350)

30.12.133 GetModuleName

Synopsis: Return the name of the current module

Declaration:
```
function GetModuleName(Module: HMODULE) : String
```

Visibility: default

Description: `GetModuleName` returns the name of the current module. On windows, this is the name of the executable when executed in an executable, or the name of the library when executed in a library.

On all other platforms, the result is always empty, since they provide no such functionality.

30.12.134 GetTempDir

Synopsis: Return name of system's temporary directory

Declaration:
```
function GetTempDir(Global: Boolean) : String
function GetTempDir : String
```

Visibility: default

Description: `GetTempDir` returns the temporary directory of the system. If `Global` is `True` (the default value) it returns the system temporary directory, if it is `False` then a directory private to the user is returned. The returned name will end with a directory delimiter character.

These directories may be the same. No guarantee is made that this directory exists or is writeable by the user.

The OnGetTempDir (1369) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: OnGetTempDir (1369), GetTempFileName (1440)

30.12.135 GetTempFileName

Synopsis: Return the name of a temporary file.

Declaration:
```
function GetTempFileName(const Dir: String;const Prefix: String)
                                : String
function GetTempFileName : String
function GetTempFileName(Dir: PChar;Prefix: PChar;uUnique: DWORD;
                         TempFileName: PChar) : DWORD
```

Visibility: default

Description: `GetTempFileName` returns the name of a temporary file in directory `Dir`. The name of the file starts with `Prefix`.

If `Dir` is empty, the value returned by `GetTempDir` is used, and if `Prefix` is empty, 'TMP' is used.

The OnGetTempFile (1369) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: GetTempDir (1440), OnGetTempFile (1369)

30.12.136 GetUserDir

Synopsis: Returns the current user's home directory.

Declaration: `function GetUserDir : String`

Visibility: default

Description: `GetUserDir` returns the home directory of the current user. On Unix-like systems (that includes Mac OS X), this is the value of the HOME environment variable. On Windows, this is the PROFILE special folder. On all other platforms, the application installation directory is returned.

If non-empty, it contains a trailing path delimiter.

See also: GetAppConfigDir (1436)

30.12.137 GUIDToString

Synopsis: Convert a `TGUID` to a string representation.

Declaration: `function GUIDToString(const GUID: TGUID) : String`

Visibility: default

Description: `GUIDToString` converts the GUID identifier in `GUID` to a string representation in the form

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where each `X` is a hexadecimal digit.

Errors: None.

See also: Supports (1477), #rtl.system.TGUID (1193), StringToGUID (1461), IsEqualGuid (1444)

30.12.138 HookSignal

Synopsis: Hook a specified signal

Declaration: `procedure HookSignal(RtlSigNum: Integer)`

Visibility: default

Description: This function is declared for Kylix compatibility, but is not implemented.

30.12.139 IncAMonth

Synopsis: Increase a date with a certain amount of months

Declaration:
```
procedure IncAMonth(var Year: Word;var Month: Word;var Day: Word;
                    NumberOfMonths: Integer)
```

Visibility: default

Description: `IncAMonth` increases the date as specified by `Year`, `Month`, `Day` with `NumberOfMonths`. It takes care of the number of days in a month when calculating the result.

This function does the same as IncMonth (1442), but operates on an already decoded date.

See also: IncMonth (1442)

30.12.140 IncludeTrailingBackslash

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration: `function IncludeTrailingBackslash(const Path: String) : String`

Visibility: default

Description: `IncludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use IncludeTrailingPathDelimiter (1442) instead.

See also: IncludeTrailingPathDelimiter (1442), ExcludeTrailingPathDelimiter (1406), PathDelim (1358), IsPathDelimiter (1445)

30.12.141 IncludeTrailingPathDelimiter

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration: `function IncludeTrailingPathDelimiter(const Path: String) : String`

Visibility: default

Description: `IncludeTrailingPathDelimiter` adds a trailing path delimiter character (PathDelim (1358)) to `Path` if none is present yet, and returns the result.

If `Path` is empty, a path delimiter is returned, for Delphi compatibility.

See also: IncludeTrailingBackslash (1442), ExcludeTrailingPathDelimiter (1406), PathDelim (1358), IsPathDelimiter (1445)

30.12.142 IncMonth

Synopsis: Increases the month in a `TDateTime` value with a given amount.

Declaration: `function IncMonth(const DateTime: TDateTime;NumberOfMonths: Integer) : TDateTime`

Visibility: default

Description: `IncMonth` increases the month number in `DateTime` with `NumberOfMonths`. It wraps the result as to get a month between 1 and 12, and updates the year accordingly. `NumberOfMonths` can be negative, and can be larger than 12 (in absolute value).

Errors: None.

See also: Date (1394), Time (1479), Now (1449)

Listing: ./sysutex/ex15.pp

```
Program Example15;

{ This program demonstrates the IncMonth function }

Uses sysutils;

Var ThisDay : TDateTime;

Begin
  ThisDay:=Date;
  Writeln ('ThisDay : ',DateToStr(ThisDay));
```

```
  WriteLn ('6 months ago :',DateToStr(IncMonth(ThisDay,-6)));
  WriteLn ('6 months from now :' ,DateToStr(IncMonth(ThisDay,6)));
  WriteLn ('12 months ago :',DateToStr(IncMonth(ThisDay,-12)));
  WriteLn ('12 months from now :' ,DateToStr(IncMonth(ThisDay,12)));
  WriteLn ('18 months ago :',DateToStr(IncMonth(ThisDay,-18)));
  WriteLn ('18 months from now :' ,DateToStr(IncMonth(ThisDay,18)));
End.
```

30.12.143 InquireSignal

Synopsis: Check whether a signal handler is set

Declaration:
```
function InquireSignal(RtlSigNum: Integer) : TSignalState
```

Visibility: default

Description: This function is declared for Kylix compatibility, but is not implemented.

30.12.144 IntToHex

Synopsis: Convert an integer value to a hexadecimal string.

Declaration:
```
function IntToHex(Value: Integer;Digits: Integer) : String
function IntToHex(Value: Int64;Digits: Integer) : String
function IntToHex(Value: QWord;Digits: Integer) : String
```

Visibility: default

Description: `IntToHex` converts `Value` to a hexadecimal string representation. The result will contain at least `Digits` characters. If `Digits` is less than the needed number of characters, the string will NOT be truncated. If `Digits` is larger than the needed number of characters, the result is padded with zeroes.

Errors: None.

See also: IntToStr (1444)

Listing: ./sysutex/ex73.pp

```
Program Example73;

{ This program demonstrates the IntToHex function }

Uses sysutils;

Var I : longint;

Begin
  For I:=0 to 31 do
      begin
      WriteLn (IntToHex(1 shl I,8));
      WriteLn (IntToHex(15 shl I,8))
      end;
End.
```

30.12.145 IntToStr

Synopsis: Convert an integer value to a decimal string.

Declaration:
```
function IntToStr(Value: Integer) : String
function IntToStr(Value: Int64) : String
function IntToStr(Value: QWord) : String
```

Visibility: default

Description: `IntToStr` coverts `Value` to it's string representation. The resulting string has only as much characters as needed to represent the value. If the value is negative a minus sign is prepended to the string.

Errors: None.

See also: IntToHex (1443), StrToInt (1474)

Listing: ./sysutex/ex74.pp

```
Program Example74;

{ This program demonstrates the IntToStr function }

Uses sysutils;

Var I : longint;

Begin
  For I:=0 to 31 do
      begin
      WriteIn (IntToStr(1 shl I));
      WriteIn (IntToStr(15 shl I));
      end;
End.
```

30.12.146 IsDelimiter

Synopsis: Check whether a given string is a delimiter character.

Declaration:
```
function IsDelimiter(const Delimiters: String;const S: String;
                     Index: Integer) : Boolean
```

Visibility: default

Description: `IsDelimiter` checks whether the `Index`-th character in the string `S` is a delimiter character as passed in `Delimiters`. If `Index` is out of range, `False` is returned.

Errors: None.

See also: LastDelimiter (1446)

30.12.147 IsEqualGUID

Synopsis: Check whether two `TGUID` variables are equal.

Declaration:
```
function IsEqualGUID(const guid1: TGUID;const guid2: TGUID) : Boolean
```

Visibility: default

Description: `IsEqualGUID` checks whether `guid1` and `guid2` are equal, and returns `True` if this is the case, or `False` otherwise.

Errors:

See also: Supports (1477), #rtl.system.TGUID (1193), StringToGUID (1461), GuidToString (1441)

30.12.148 IsLeapYear

Synopsis: Determine whether a year is a leap year.

Declaration: `function IsLeapYear(Year: Word) : Boolean`

Visibility: default

Description: `IsLeapYear` returns `True` if `Year` is a leap year, `False` otherwise.

Errors: None.

See also: IncMonth (1442), Date (1394)

Listing: ./sysutex/ex16.pp

```
Program Example16;

{ This program demonstrates the IsLeapYear function }

Uses sysutils;

Var YY,MM,dd : Word;

Procedure TestYear (Y : Word);

begin
  WriteLn (Y,' is leap year : ',IsLeapYear(Y));
end;

Begin
  DeCodeDate(Date,YY,mm,dd);
  TestYear(yy);
  TestYear(2000);
  TestYear(1900);
  TestYear(1600);
  TestYear(1992);
  TestYear(1995);
End.
```

30.12.149 IsPathDelimiter

Synopsis: Is the character at the given position a pathdelimiter ?

Declaration: `function IsPathDelimiter(const Path: String;Index: Integer) : Boolean`

Visibility: default

Description: `IsPathDelimiter` returns `True` if the character at position `Index` equals PathDelim (1358), i.e. if it is a path delimiter character for the current platform.

Errors: IncludeTrailingPathDelimiter (1442)ExcludeTrailingPathDelimiter (1406)PathDelim (1358)

30.12.150 IsValidIdent

Synopsis: Check whether a string is a valid identifier name.

Declaration: `function IsValidIdent(const Ident: String) : Boolean`

Visibility: default

Description: `IsValidIdent` returns `True` if `Ident` can be used as a compoent name. It returns `False` otherwise. `Ident` must consist of a letter or underscore, followed by a combination of letters, numbers or underscores to be a valid identifier.

Errors: None.

Listing: ./sysutex/ex75.pp

```
Program Example75;

{ This program demonstrates the IsValidIdent function }

Uses sysutils;

Procedure Testit (S : String);

begin
  Write ('"',S,'" is ');
  If not IsVAlidIdent(S) then
    Write('NOT ');
  WriteIn ('a valid identifier');
end;

Begin
  Testit ('_MyObj');
  Testit ('My__Obj1');
  Testit ('My_1_Obj');
  Testit ('1MyObject');
  Testit ('My@Object');
  Testit ('M123');
End.
```

30.12.151 LastDelimiter

Synopsis: Return the last occurrence of a set of delimiters in a string.

Declaration:
```
function LastDelimiter(const Delimiters: String;const S: String)
                       : Integer
```

Visibility: default

Description: `LastDelimiter` returns the *last* occurrence of any character in the set `Delimiters` in the string `S`.

Errors:

Listing: ./sysutex/ex88.pp

```
Program example88;

{ This program demonstrates the LastDelimiter function }

uses SysUtils;

begin
  WriteLn(LastDelimiter('\.:','c:\filename.ext'));
end.
```

30.12.152 LeftStr

Synopsis: Return a number of characters starting at the left of a string.

Declaration: `function LeftStr(const S: String;Count: Integer) : String`

Visibility: default

Description: `LeftStr` returns the `Count` leftmost characters of `S`. It is equivalent to a call to `Copy(S,1,Count)`.

Errors: None.

See also: RightStr (1452), TrimLeft (1482), TrimRight (1483), Trim (1481)

Listing: ./sysutex/ex76.pp

```
Program Example76;

{ This program demonstrates the LeftStr function }

Uses sysutils;

Begin
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',20));
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',15));
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',1));
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',200));
End.
```

30.12.153 LoadStr

Synopsis: Load a string from the resource tables.

Declaration: `function LoadStr(Ident: Integer) : String`

Visibility: default

Description: This function is not yet implemented. resources are not yet supported.

Errors:

30.12.154 LowerCase

Synopsis: Return a lowercase version of a string.

Declaration:
```
function LowerCase(const s: String) : String;  Overload
function LowerCase(const V: variant) : String;  Overload
```

Visibility: default

Description: `LowerCase` returns the lowercase equivalent of `S`. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the lowercase function of the system unit, and is provided for compatiibility only.

Errors: None.

See also: AnsiLowerCase (1376), UpperCase (1487), AnsiUpperCase (1384)

Listing: ./sysutex/ex77.pp

```
Program Example77;

{ This program demonstrates the LowerCase function }

Uses sysutils;

Begin
  WriteIn (LowerCase('THIS WILL COME out all LoWeRcAsE !'));
End.
```

30.12.155 MSecsToTimeStamp

Synopsis: Convert a number of milliseconds to a `TDateTime` value.

Declaration:
```
function MSecsToTimeStamp(MSecs: Comp) : TTimeStamp
```

Visibility: default

Description: `MSecsTiTimeStamp` converts the given number of milliseconds to a `TTimeStamp` date/time notation.

Use `TTimeStamp` variables if you need to keep very precise track of time.

Errors: None.

See also: TimeStampToMSecs (1481), DateTimeToTimeStamp (1397)

Listing: ./sysutex/ex17.pp

```
Program Example17;

{ This program demonstrates the MSecsToTimeStamp function }

Uses sysutils;

Var MS : Comp;
    TS : TTimeStamp;
    DT : TDateTime;
Begin
  TS:=DateTimeToTimeStamp(Now);
```

```
  WriteLn ('Now in days since 1/1/0001      : ',TS.Date);
  WriteLn ('Now in millisecs since midnight : ',TS.Time);
  MS:=TimeStampToMSecs(TS);
  WriteLn ('Now in millisecs since 1/1/0001 : ',MS);
  MS:=MS-1000*3600*2;
  TS:=MSecsToTimeStamp(MS);
  DT:=TimeStampToDateTime(TS);
  WriteLn ('Now minus 1 day : ',DateTimeToStr(DT));
End.
```

30.12.156 NewStr

Synopsis: Allocate a new ansistring on the heap.

Declaration: `function NewStr(const S: String) : PString`

Visibility: default

Description: `NewStr` assigns a new dynamic string on the heap, copies `S` into it, and returns a pointer to the newly assigned string.

This function is obsolete, and shouldn't be used any more. The `AnsiString` mechanism also allocates ansistrings on the heap, and should be preferred over this mechanism.

For an example, see AssignStr (1386).

Errors: If not enough memory is present, an EOutOfMemory exception will be raised.

See also: AssignStr (1386), DisposeStr (1402)

30.12.157 Now

Synopsis: Returns the current date and time.

Declaration: `function Now : TDateTime`

Visibility: default

Description: `Now` returns the current date and time. It is equivalent to `Date+Time`.

Errors: None.

See also: Date (1394), Time (1479)

Listing: ./sysutex/ex18.pp

```
Program Example18;

{ This program demonstrates the Now function }

Uses sysutils;

Begin
  WriteLn ('Now : ',DateTimeToStr(Now));
End.
```

30.12.158 OutOfMemoryError

Synopsis: Raise an `EOutOfMemory` exception

Declaration: `procedure OutOfMemoryError`

Visibility: default

Description: `OutOfMemoryError` raises an EOutOfMemory (1494) exception, with an exception object that has been allocated on the heap at program startup. The program should neved create an EOutOfMemory (1494) exception, but always call this routine.

See also: EOutOfMemory (1494)

30.12.159 QuotedStr

Synopsis: Return a quotes version of a string.

Declaration: `function QuotedStr(const S: String) : String`

Visibility: default

Description: `QuotedStr` returns the string `S`, quoted with single quotes. This means that `S` is enclosed in single quotes, and every single quote in `S` is doubled. It is equivalent to a call to `AnsiQuotedStr(s, "")`.

Errors: None.

See also: AnsiQuotedStr (1377), AnsiExtractQuotedStr (1375)

Listing: ./sysutex/ex78.pp

```
Program Example78;

{ This program demonstrates the QuotedStr function }

Uses sysutils;

Var S : AnsiString;

Begin
  S:='He said ''Hello'' and walked on';
  Writeln (S);
  Writeln ('  becomes');
  Writeln (QuotedStr(S));
End.
```

30.12.160 RaiseLastOSError

Synopsis: Raise an exception with the last Operating System error code.

Declaration: `procedure RaiseLastOSError`

Visibility: default

Description: `RaiseLastOSError` raises an EOSError (1494) exception with the error code returned by `GetLastOSError`. If the Error code is nonzero, then the corresponding error message will be returned. If the error code is zero, a standard message will be returned.

Errors: This procedure may not be implemented on all platforms. If it is not, then a normal Exception (1496) will be raised.

See also: EOSError (1494), GetLastOSError (1439), Exception (1496)

30.12.161 RemoveDir

Synopsis: Remove a directory from the filesystem.

Declaration: `function RemoveDir(const Dir: String) : Boolean`

Visibility: default

Description: `RemoveDir` removes directory `Dir` from the disk. If the directory is not absolue, it is appended to the current working directory.

For an example, see CreateDir (1392).

Errors: In case of error (e.g. the directory isn't empty) the function returns `False`. If successful, `True` is returned.

30.12.162 RenameFile

Synopsis: Rename a file.

Declaration: `function RenameFile(const OldName: String;const NewName: String) : Boolean`

Visibility: default

Description: `RenameFile` renames a file from `OldName` to `NewName`. The function returns `True` if successful, `False` otherwise. *Remark:* you cannot rename across disks or partitions.

Errors: On Error, `False` is returned.

See also: DeleteFile (1400)

Listing: ./sysutex/ex44.pp

```
Program Example44;

{ This program demonstrates the RenameFile function }

Uses sysutils;

Var F : Longint;
    S : String;

Begin
  S:='Some short file.';
  F:=FileCreate ('test.dap');
  FileWrite(F,S[1],Length(S));
  FileClose(F);
  If RenameFile ('test.dap','test.dat') then
    WriteIn ('Successfully renamed files.');
End.
```

30.12.163 ReplaceDate

Synopsis: Replace the date part of a date/time stamp

Declaration: `procedure ReplaceDate(var DateTime: TDateTime;const NewDate: TDateTime)`

Visibility: default

Description: `ReplaceDate` replaces the date part of `DateTime` with `NewDate`. The time part is left unchanged.

See also: ReplaceTime (1452)

30.12.164 ReplaceTime

Synopsis: Replace the time part

Declaration: `procedure ReplaceTime(var dati: TDateTime;NewTime: TDateTime)`

Visibility: default

Description: `ReplaceTime` replaces the time part in `dati` with `NewTime`. The date part remains untouched.

Errors:

30.12.165 RightStr

Synopsis: Return a number of characters from a string, starting at the end.

Declaration: `function RightStr(const S: String;Count: Integer) : String`

Visibility: default

Description: `RightStr` returns the `Count` rightmost characters of `S`. It is equivalent to a call to `Copy(S,Length(S)+1-Count,Coun`

If `Count` is larger than the actual length of `S` only the real length will be used.

Errors: None.

See also: LeftStr (1447), Trim (1481), TrimLeft (1482), TrimRight (1483)

Listing: ./sysutex/ex79.pp

```
Program Example79;

{ This program demonstrates the RightStr function }

Uses sysutils;

Begin
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',20));
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',15));
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',1));
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',200));
End.
```

30.12.166 SafeLoadLibrary

Synopsis: Load a library safely

Declaration:
```
function SafeLoadLibrary(const FileName: AnsiString;ErrorMode: DWord)
                                 : HMODULE
```

Visibility: default

Description: `SafeLoadLibrary` saves and restores some registers before and after issuing a call to `LoadLibrary`.

Errors: None.

30.12.167 SameFileName

Synopsis: Are two filenames referring to the same file ?

Declaration:
```
function SameFileName(const S1: String;const S2: String) : Boolean
```

Visibility: default

Description: `SameFileName` returns `True` if calling AnsiCompareFileName (1372) with arguments `S1` and `S2` returns 0, and returns `False` otherwise.

Errors: None.

See also: AnsiCompareFileName (1372)

30.12.168 SameText

Synopsis: Checks whether 2 strings are the same (case insensitive)

Declaration:
```
function SameText(const s1: String;const s2: String) : Boolean
```

Visibility: default

Description: `SameText` calls CompareText (1391) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

Errors: None.

See also: CompareText (1391), AnsiSameText (1378), AnsiSameStr (1378)

30.12.169 SetCurrentDir

Synopsis: Set the current directory of the application.

Declaration:
```
function SetCurrentDir(const NewDir: String) : Boolean
```

Visibility: default

Description: `SetCurrentDir` sets the current working directory of your program to `NewDir`. It returns `True` if the function was successfull, `False` otherwise.

Errors: In case of error, `False` is returned.

See also: GetCurrentDir (1437)

30.12.170 SetDirSeparators

Synopsis: Set the directory separators to the known directory separators.

Declaration: `function SetDirSeparators(const FileName: String) : String`

Visibility: default

Description: `SetDirSeparators` returns `FileName` with all possible DirSeparators replaced by `OSDirSeparator`.

Errors: None.

See also: ExpandFileName (1406), ExtractFilePath (1409), ExtractFileDir (1407)

Listing: ./sysutex/ex47.pp

```
Program Example47;

{ This program demonstrates the SetDirSeparators function }

Uses sysutils;

Begin
  WriteLn (SetDirSeparators('/pp\bin/win32\ppc386'));
End.
```

30.12.171 ShowException

Synopsis: Show the current exception to the user.

Declaration: `procedure ShowException(ExceptObject: TObject;ExceptAddr: Pointer)`

Visibility: default

Description: `ShowException` shows a message stating that a `ExceptObject` was raised at address `ExceptAddr`. It uses ExceptionErrorMessage (1405) to create the message, and is aware of the fact whether the application is a console application or a GUI application. For a console application, the message is written to standard error output. For a GUI application, OnShowException (1369) is executed.

Errors: If, for a GUI application, OnShowException (1369) is not set, no message will be displayed to the user.

The exception message can be at most 255 characters long: It is possible that no memory can be allocated on the heap, so ansistrings are not available, so a shortstring is used to display the message.

See also: ExceptObject (1405), ExceptAddr (1404), ExceptionErrorMessage (1405)

30.12.172 Sleep

Synopsis: Suspend execution of a program for a certain time.

Declaration: `procedure Sleep(milliseconds: Cardinal)`

Visibility: default

Description: `Sleep` suspends the execution of the program for the specified number of milliseconds (`milliseconds`). After the specified period has expired, program execution resumes.

Remark: The indicated time is not exact, i.e. it is a minimum time. No guarantees are made as to the exact duration of the suspension.

30.12.173 SScanf

Synopsis: Scan a string for substrings and return the substrings

Declaration:
```
function SScanf(const s: String;const fmt: String;
                const Pointers: Array of Pointer) : Integer
```

Visibility: default

Description: SSCanF scans the string S for the elements specified in Fmt, and returns the elements in the pointers in Pointers. The Fmt can contain placeholders of the form %X where X can be one of the following characters:

dPlaceholder for a decimal number.

fPlaceholder for a floating point number (an extended)

sPlaceholder for a string of arbitrary length.

cPlaceholder for a single character

The Pointers array contains a list of pointers, each pointer should point to a memory location of a type that corresponds to the type of placeholder in that position:

dA pointer to an integer.

fA pointer to an extended.

sA pointer to an ansistring.

cA pointer to a single character.

Errors: No error checking is performed on the type of the memory location.

See also: Format (1427)

30.12.174 StrAlloc

Synopsis: Allocate a null-terminated string on the heap.

Declaration:
```
function StrAlloc(Size: cardinal) : PChar
```

Visibility: default

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

Additionally, StrAlloc allocates 4 extra bytes to store the size of the allocated memory. Therefore this function is NOT compatible with the StrAlloc (1117) function of the Strings unit.

For an example, see StrBufSize (1456).

Errors: None.

See also: StrBufSize (1456), StrDispose (1458), #rtl.strings.StrAlloc (1117)

30.12.175 StrBufSize

Synopsis: Return the size of a null-terminated string allocated on the heap.

Declaration: `function StrBufSize(Str: PChar) : SizeUInt`

Visibility: default

Description: `StrBufSize` returns the memory allocated for `Str`. This function ONLY gives the correct result if `Str` was allocated using StrAlloc (1455).

Errors: If no more memory is available, a runtime error occurs.

See also: StrAlloc (1455), StrDispose (1458)

Listing: ./sysutex/ex46.pp

```
Program Example46;

{ This program demonstrates the StrBufSize function }
{$H+}

Uses sysutils;

Const S = 'Some nice string';

Var P : Pchar;

Begin
  P:=StrAlloc(Length(S)+1);
  StrPCopy(P,S);
  Write (P, ' has length ',length(S));
  WriteIn (' and  buffer size ',StrBufSize(P));
  StrDispose(P);
End.
```

30.12.176 StrByteType

Synopsis: Return the type of byte in a null-terminated string for a multi-byte character set

Declaration: `function StrByteType(Str: PChar;Index: Cardinal) : TMbcsByteType`

Visibility: default

Description: `StrByteType` returns the type of byte in the null-terminated string `Str` at (0-based) position `Index`.

Errors: No checking on the index is performed.

See also: TMbcsByteType (1365), ByteType (1388)

30.12.177 strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function strcat(dest: pchar;source: pchar) : pchar`

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: StrLCat (1462)

Listing: ./stringex/ex11.pp

```
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 }
  WriteIn ('P2 : ',P2);
  StrDispose(P2);
end.
```

30.12.178 StrCharLength

Synopsis: Return the length of a null-terminated string in characters.

Declaration: `function StrCharLength(const Str: PChar) : Integer`

Visibility: default

Description: `StrCharLength` returns the length of the null-terminated string `Str` (a widestring) in characters (not in bytes). It uses the widestring manager to do this.

30.12.179 strcomp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration: `function strcomp(str1: pchar;str2: pchar) : SizeInt`

Visibility: default

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`.

For an example, see StrLComp (1462).

Errors: None.

See also: StrLComp (1462), StrIComp (1460), StrLIComp (1465)

30.12.180 strcopy

Synopsis: Copy a null-terminated string

Declaration: `function strcopy(dest: pchar;source: pchar) : pchar`

Visibility: default

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 (1467), StrLCopy (1463), StrECopy (1459)

Listing: ./stringex/ex4.pp

```
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
    WriteIn ('Oh-oh problems...')
  else
    WriteIn ('All is well : PP=',PP);
  StrDispose(PP);
end.
```

30.12.181 StrDispose

Synopsis: Dispose of a null-terminated string on the heap.

Declaration: `procedure StrDispose(Str: PChar)`

Visibility: default

Description: `StrDispose` frees any memory allocated for `Str`. This function will only function correctly if `Str` has been allocated using StrAlloc (1455) from the SysUtils unit.

For an example, see StrBufSize (1456).

Errors: If an invalid pointer is passed, or a pointer not allocated with `StrAlloc`, an error may occur.

See also: StrBufSize (1456), StrAlloc (1455), StrDispose (1458)

30.12.182 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strecopy(dest: pchar;source: pchar) : pchar`

Visibility: default

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 (1463), StrCopy (1458)

Listing: ./stringex/ex6.pp

```
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
    WriteIn('Something is wrong here !')
  else
    WriteIn ('PP= ',PP);
  StrDispose(PP);
end.
```

30.12.183 strend

Synopsis: Return a pointer to the end of a null-terminated string

Declaration: `function strend(p: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the end of `P`. (i.e. to the terminating null-character.

Errors: None.

See also: StrLen (1464)

Listing: ./stringex/ex7.pp

```
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
    WriteIn('Something is wrong here !')
  else
    WriteIn ('All is well..');
end.
```

30.12.184 StrFmt

Synopsis: Format a string with given arguments, store the result in a buffer.

Declaration:
```
function StrFmt(Buffer: PChar;Fmt: PChar;const args: Array of const)
                  : Pchar
function StrFmt(Buffer: PChar;Fmt: PChar;const Args: Array of const;
                const FormatSettings: TFormatSettings) : PChar
```

Visibility: default

Description: `StrFmt` will format `fmt` with `Args`, as the Format (1427) function does, and it will store the result in `Buffer`. The function returns `Buffer`. `Buffer` should point to enough space to contain the whole result.

Errors: for a list of errors, see Format (1427).

See also: StrLFmt (1464), FmtStr (1426), Format (1427), FormatBuf (1432)

Listing: ./sysutex/ex80.pp

```
Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin
  SetLEngth(S,80);
  WriteIn (StrFmt (@S[1],'For some nice examples of fomatting see %s.',['Format']));
End.
```

30.12.185 stricomp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration:
```
function stricomp(str1: pchar;str2: pchar) : SizeInt
```

Visibility: default

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 (1462), StrComp (1457), StrLIComp (1465)

Listing: ./stringex/ex8.pp

```
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.
```

30.12.186 StringReplace

Synopsis: Replace occurrences of one substring with another in a string.

Declaration:
```
function StringReplace(const S: String;const OldPattern: String;
                       const NewPattern: String;Flags: TReplaceFlags)
                        : String
```

Visibility: default

Description: `StringReplace` searches the string `S` for occurrences of the string `OldPattern` and, if it is found, replaces it with `NewPattern`. It returns the resulting string. The behaviour of `StringReplace` can be runed with `Flags`, which is of type TReplaceFlags (1366). Standard behaviour is to replace only the first occurrence of `OldPattern`, and to search case sensitively.

Errors: None.

See also: TReplaceFlags (1366)

30.12.187 StringToGUID

Synopsis: Convert a string to a native `TGUID` type.

Declaration:
```
function StringToGUID(const S: String) : TGUID
```

Visibility: default

Description: `StringToGUID` converts the string `S` to a valid GUID. The string `S` should be of the form

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where each X is a hexadecimal digit. The dashes and braces are required.

Errors: In case `S` contains an invalid GUID representation, a EConvertError (1492) exception is raised.

See also: Supports (1477), #rtl.system.TGUID (1193), GUIDToString (1441), IsEqualGuid (1444)

30.12.188 strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function strlcat(dest: pchar;source: pchar;l: SizeInt) : pchar`

Visibility: default

Description: Adds `MaxLen` characters from `Source` to `Dest`, and adds a terminating null-character. Returns `Dest`.

Errors: None.

See also: StrCat (1456)

Listing: ./stringex/ex12.pp

```
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);
  WriteIn ('P2 = ',P2);
  StrDispose(P2)
end.
```

30.12.189 strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings

Declaration: `function strlcomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

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 (1457), StrIComp (1460), StrLIComp (1465)

Listing: ./stringex/ex8.pp

```
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.
```

30.12.190 strlcopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: `function strlcopy(dest: pchar;source: pchar;maxlen: SizeInt) : pchar`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`, and makes `Dest` a null terminated string.

Errors: No length checking is performed.

See also: StrCopy (1458), StrECopy (1459)

Listing: ./stringex/ex5.pp

```
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));
  StrDispose(PP);
end.
```

30.12.191 strlen

Synopsis: Length of a null-terminated string.

Declaration:
```
function strlen(p: pchar) : sizeint
```

Visibility: default

Description: Returns the length of the null-terminated string `P`.

Errors: None.

See also: StrNew (1466)

Listing: ./stringex/ex1.pp

```
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.
```

30.12.192 StrLFmt

Synopsis: Format a string with given arguments, but with limited length.

Declaration:
```
function StrLFmt(Buffer: PCHar;Maxlen: Cardinal;Fmt: PChar;
                 const args: Array of const) : Pchar
function StrLFmt(Buffer: PCHar;Maxlen: Cardinal;Fmt: PChar;
                 const args: Array of const;
                 const FormatSettings: TFormatSettings) : Pchar
```

Visibility: default

Description: `StrLFmt` will format `fmt` with `Args`, as the Format (1427) function does, and it will store maximally `Maxlen characters` of the result in `Buffer`. The function returns `Buffer`. `Buffer` should point to enough space to contain `MaxLen` characters.

Errors: for a list of errors, see Format (1427).

See also: StrFmt (1460), FmtStr (1426), Format (1427), FormatBuf (1432)

Listing: ./sysutex/ex81.pp

```
Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;
```

```
Begin
  SetLEngth(S,80);
  WriteLn (StrLFmt (@S[1],80,'For some nice examples of fomatting see %s.',['Format']));
End.
```

30.12.193 strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration: `function strlicomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

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`.

For an example, see StrIComp (1460)

Errors: None.

See also: StrLComp (1462), StrComp (1457), StrIComp (1460)

30.12.194 strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration: `function strlower(p: pchar) : pchar`

Visibility: default

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

Errors: None.

See also: StrUpper (1477)

Listing: ./stringex/ex14.pp

```
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.
```

30.12.195 strmove

Synopsis: Move a null-terminated string to new location.

Declaration: `function strmove(dest: pchar;source: pchar;l: SizeInt) : pchar`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: StrLCopy (1463), StrCopy (1458)

Listing: ./stringex/ex10.pp

```
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 }
  WriteIn ('P2 = ',P2);
  StrDispose(P2);
end.
```

30.12.196 strnew

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: pchar) : pchar`

Visibility: default

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: StrCopy (1458), StrDispose (1458)

Listing: ./stringex/ex16.pp

```
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);
  StrDispose(P2);
end.
```

30.12.197 StrNextChar

Synopsis: Returns a pointer to the location of the next empty character in a null-terminated string

Declaration: `function StrNextChar(const Str: PChar) : PChar`

Visibility: default

Description: `StrNextChar` returns a pointer to the null-character that terminates the string `Str`

Errors: if `Str` is not properly terminated, an access violation may occur.

30.12.198 StrPas

Synopsis: Convert a null-terminated string to an ansistring.

Declaration: `function StrPas(Str: PChar) : String`

Visibility: default

Description: Converts a null terminated string in `Str` to an Ansitring, and returns this string. This string is NOT truncated at 255 characters as is the

Errors: None.

See also: StrPCopy (1467), StrPLCopy (1468)

30.12.199 StrPCopy

Synopsis: Copy an ansistring to a null-terminated string.

Declaration: `function StrPCopy(Dest: PChar;Source: String) : PChar`

Visibility: default

Description: `StrPCopy` Converts the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the string `Source`, i.e. `Length(Source)+1` bytes.

Errors: No checking is performed to see whether `Dest` points to enough memory to contain `Source`.

See also: StrPLCopy (1468), StrPas (1467)

30.12.200 StrPLCopy

Synopsis: Copy a limited number of characters from an ansistring to a null-terminated string.

Declaration: `function StrPLCopy(Dest: PChar;Source: String;MaxLen: SizeUInt) : PChar`

Visibility: default

Description: `StrPLCopy` Converts maximally `MaxLen` characters of the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the characters.

Errors: No checking is performed to see whether `Dest` points to enough memory to contain L characters of `Source`.

See also: StrPCopy (1467)

30.12.201 strpos

Synopsis: Find position of one null-terminated substring in another.

Declaration: `function strpos(str1: pchar;str2: pchar) : pchar`

Visibility: default

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: StrScan (1469), StrRScan (1468)

Listing: ./stringex/ex15.pp

```
Program Example15;

Uses strings;

{ Program to demonstrate the StrPos function. }

Const P : PChar = 'This is a PChar string.';
      S : Pchar = 'is';
begin
  WriteIn ('Position of ''is'' in P : ',longint(StrPos(P,S))-Longint(P));
end.
```

30.12.202 strrscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function strrscan(p: pchar;c: Char) : pchar`

Visibility: default

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`.

For an example, see StrScan (1469).

Errors: None.

See also: StrScan (1469), StrPos (1468)

30.12.203 strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: pchar;c: Char) : pchar`

Visibility: default

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: StrRScan (1468), StrPos (1468)

Listing: ./stringex/ex13.pp

```
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.
```

30.12.204 StrToBool

Synopsis: Convert a string to a boolean value

Declaration: `function StrToBool(const S: String) : Boolean`

Visibility: default

Description: `StrToBool` will convert the string `S` to a boolean value. The string `S` can contain one of `'True'`, `'False'` (case is ignored) or a numerical value. If it contains a numerical value, 0 is converted to `False`, all other values result in `True`. If the string `S` contains no valid boolean, then an EConvertError (1492) exception is raised.

Errors: On error, an EConvertError (1492) exception is raised.

See also: BoolToStr (1387)

30.12.205 StrToBoolDef

Synopsis: Convert string to boolean value, returning default in case of error

Declaration: `function StrToBoolDef(const S: String;Default: Boolean) : Boolean`

Visibility: default

Description: `StrToBoolDef` tries to convert the string `S` to a boolean value, and returns the boolean value in case of success. In case `S` does not contain a valid boolean string, `Default` is returned.

See also: StrToBool (1469), TryStrToBool (1484)

30.12.206 StrToCurr

Synopsis: Convert a string to a currency value

Declaration: `function StrToCurr(const S: String) : Currency`

Visibility: default

Description: `StrToCurr` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, an EConvertError (1492) exception is raised.

Errors: On error, an EConvertError (1492) exception is raised.

See also: CurrToStr (1393), StrToCurrDef (1470)

30.12.207 StrToCurrDef

Synopsis: Convert a string to a currency value, using a default value

Declaration: `function StrToCurrDef(const S: String;Default: Currency) : Currency`

Visibility: default

Description: `StrToCurrDef` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, the fallback `Default` value is returned.

Errors: On error, the `Default` value is returned.

See also: CurrToStr (1393), StrToCurr (1470)

30.12.208 StrToDate

Synopsis: Convert a date string to a `TDateTime` value.

Declaration: `function StrToDate(const S: String) : TDateTime`

Visibility: default

Description: `StrToDate` converts the string `S` to a `TDateTime` date value. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: StrToTime (1476), DateToStr (1397), TimeToStr (1481)

Listing: ./sysutex/ex19.pp

```
Program Example19;

{ This program demonstrates the StrToDate function }

Uses sysutils;

Procedure TestStr (S : String);
```

```
begin
  WriteIn (S,' : ',DateToStr(StrToDate(S)));
end;

Begin

  WriteIn ('ShortDateFormat ',ShortDateFormat);
  TestStr(DateTimeToStr(Date));
  TestStr('05/05/1999');
  TestStr('5/5');
  TestStr('5');
End.
```

30.12.209 StrToDateDef

Synopsis: Convert string to date, returning a default value

Declaration:
```
function StrToDateDef(const S: String;const Defvalue: TDateTime)
                          : TDateTime
```

Visibility: default

Description: `StrToDateDef` tries to convert the string `S` to a valid `TDateTime` date value, and returns `DefValue` if `S` does not contain a valid date indication.

Errors: None.

See also: StrToDate (1470), TryStrToDate (1485), StrToTimeDef (1477)

30.12.210 StrToDateTime

Synopsis: Convert a date/time string to a `TDateTime` value.

Declaration:
```
function StrToDateTime(const S: String) : TDateTime
```

Visibility: default

Description: `StrToDateTime` converts the string `S` to a `TDateTime` date and time value. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: StrToDate (1470), StrToTime (1476), DateTimeToStr (1395)

Listing: ./sysutex/ex20.pp

```
Program Example20;

{ This program demonstrates the StrToDateTime function }

Uses sysutils;

Procedure TestStr (S : String);
```

```
begin
  WriteIn (S,' : ',DateTimeToStr(StrToDateTime(S)));
end;

Begin

  WriteIn ('ShortDateFormat ',ShortDateFormat);
  TestStr(DateTimeToStr(Now));
  TestStr('05-05-1999 15:50');
  TestStr('5-5 13:30');
  TestStr('5 1:30PM');
End.
```

30.12.211 StrToDateTimeDef

Synopsis: Convert string to date/time, returning a default value

Declaration:
```
function StrToDateTimeDef(const S: String;const Defvalue: TDateTime)
                                  : TDateTime
```

Visibility: default

Description: `StrToDateTimeDef` tries to convert the string `S` to a valid `TDateTime` date and time value, and returns `DefValue` if `S` does not contain a valid date-time indication.

Errors: None.

See also: StrToTimeDef (1477), StrToDateDef (1471), TryStrToDateTime (1485), StrToDateTime (1471)

30.12.212 StrToFloat

Synopsis: Convert a string to a floating-point value.

Declaration:
```
function StrToFloat(const S: String) : Extended
function StrToFloat(const S: String;
                    const FormatSettings: TFormatSettings) : Extended
```

Visibility: default

Description: `StrToFloat` converts the string `S` to a floating point value. `S` should contain a valid string representation of a floating point value (either in decimal or scientific notation). The `thousandseparator` character may however not be used.

Up to and including version 2.2.2 of the compiler, if the string contains a decimal value, then the decimal separator character can either be a '.' or the value of the `DecimalSeparator` variable.

As of version 2.3.1, the string may contain only the `DecimalSeparator` character. The dot ('.') can no longer be used instead of the `DecimalSeparator`.

Errors: If the string `S` doesn't contain a valid floating point string, then an exception will be raised.

See also: TextToFloat (1478), FloatToStr (1422), FormatFloat (1434), StrToInt (1474)

Listing: ./sysutex/ex90.pp

```
Program Example90;

{ This program demonstrates the StrToFloat function }
{$mode objfpc}
{$h+ }

Uses SysUtils;

Const
  NrValues = 5;
  TestStr : Array[1..NrValues] of string =
          ('1,1','-0,2','1,2E-4','0','1E4');

Procedure Testit;

Var
  I : Integer;
  E : Extended;

begin
  Writeln('Using DecimalSeparator : ',DecimalSeparator);
  For I:=1 to NrValues do
    begin
    Writeln('Converting : ',TestStr[i]);
    Try
      E:=StrToFloat(TestStr[i]);
      Writeln('Converted value : ',E);
    except
      On E : Exception do
        Writeln('Exception when converting : ',E.Message);
    end;
    end;
end;

Begin
  DecimalSeparator:=',';
  Testit;
  DecimalSeparator:='.';
  Testit;
End.
```

30.12.213 StrToFloatDef

Synopsis: Convert a string to a float, with a default value.

Declaration:
```
function StrToFloatDef(const S: String;const Default: Extended)
                            : Extended
function StrToFloatDef(const S: String;const Default: Extended;
                       const FormatSettings: TFormatSettings) : Extended
```

Visibility: default

Description: `StrToFloatDef` tries to convert the string `S` to a floating point value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

30.12.214 StrToInt

Synopsis: Convert a string to an integer value.

Declaration: `function StrToInt(const s: String) : Integer`

Visibility: default

Description: `StrToInt` will convert the string `S`to an integer. If the string contains invalid characters or has an invalid format, then an `EConvertError` is raised.

To be successfully converted, a string can contain a combination of `numerical` characters, possibly preceded by a minus sign (-). Spaces are not allowed.

Errors: In case of error, an `EConvertError` is raised.

See also: IntToStr (1444), StrToIntDef (1475)

Listing: ./sysutex/ex82.pp

```
Program Example82;

{$mode objfpc}

{ This program demonstrates the StrToInt function }

Uses sysutils;

Begin
  WriteLn (StrToInt('1234'));
  WriteLn (StrToInt('-1234'));
  WriteLn (StrToInt('0'));
  Try
    WriteLn (StrToInt('12345678901234567890'));
  except
    On E : EConvertError do
      WriteLn ('Invalid number encountered');
  end;
End.
```

30.12.215 StrToInt64

Synopsis: Convert a string to an Int64 value.

Declaration: `function StrToInt64(const s: String) : Int64`

Visibility: default

Description: `StrToInt64` converts the string `S` to a Int64 value, and returns this value. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the $ character) are supported.

Errors: On error, a EConvertError (1492) exception is raised.

See also: TryStrToInt64 (1486), StrToInt64Def (1475), StrToInt (1474), TryStrToInt (1486), StrToIntDef (1475)

30.12.216 StrToInt64Def

Synopsis: Convert a string to an Int64 value, with a default value

Declaration: `function StrToInt64Def(const S: String;Default: Int64) : Int64`

Visibility: default

Description: `StrToInt64Def` tries to convert the string `S` to a Int64 value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

See also: StrToInt64 (1474), TryStrToInt64 (1486), StrToInt (1474), TryStrToInt (1486), StrToIntDef (1475)

30.12.217 StrToIntDef

Synopsis: Convert a string to an integer value, with a default value.

Declaration: `function StrToIntDef(const S: String;Default: Integer) : Integer`

Visibility: default

Description: `StrToIntDef` will convert a string to an integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of `numerical` characters, possibly preceded by a minus sign (-). Spaces are not allowed.

Errors: None.

See also: IntToStr (1444), StrToInt (1474)

Listing: ./sysutex/ex83.pp

```
Program Example82;

{$mode objfpc}

{ This program demonstrates the StrToInt function }

Uses sysutils;

Begin
  WriteLn (StrToIntDef('1234',0));
  WriteLn (StrToIntDef('-1234',0));
  WriteLn (StrToIntDef('0',0));
  Try
    WriteLn (StrToIntDef('12345678901234567890',0));
  except
    On E : EConvertError do
      WriteLn ('Invalid number encountered');
  end;
End.
```

30.12.218 StrToQWord

Synopsis: Convert a string to a QWord.

Declaration: `function StrToQWord(const s: String) : QWord`

Visibility: default

Description: `TryStrToQWord` converts the string `S` to a valid QWord (unsigned 64-bit) value, and returns the result.

Errors: If the string `S` does not contain a valid QWord value, a EConvertError (1492) exception is raised.

See also: TryStrToQWord (1486), StrToQWordDef (1476), StrToInt64 (1474), StrToInt (1474)

30.12.219 StrToQWordDef

Synopsis: Try to convert a string to a QWord, returning a default value in case of failure.

Declaration: `function StrToQWordDef(const S: String;Default: QWord) : QWord`

Visibility: default

Description: `StrToQWordDef` tries to convert the string `S` to a valid QWord (unsigned 64-bit) value, and returns the result. If the conversion fails, the function returns the value passed in `Def`.

See also: StrToQWord (1476), TryStrToQWord (1486), StrToInt64Def (1475), StrToIntDef (1475)

30.12.220 StrToTime

Synopsis: Convert a time string to a `TDateTime` value.

Declaration: `function StrToTime(const S: String) : TDateTime`

Visibility: default

Description: `StrToTime` converts the string `S` to a `TDateTime` time value. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: StrToDate (1470), StrToDateTime (1471), TimeToStr (1481)

Listing: ./sysutex/ex21.pp

```
Program Example21;

{ This program demonstrates the StrToTime function }

Uses sysutils;

Procedure TestStr (S : String);

begin
  WriteIn (S,' : ',TimeToStr(StrToTime(S)));
end;

Begin
  teststr (TimeToStr(Time));
```

```
  teststr ('12:00');
  teststr ('15:30');
  teststr ('3:30PM');
End.
```

30.12.221 StrToTimeDef

Synopsis: Convert string to time, returning a default value

Declaration:
```
function StrToTimeDef(const S: String;const Defvalue: TDateTime)
                      : TDateTime
```

Visibility: default

Description: `StrToTimeDef` tries to convert the string `S` to a valid `TDateTime` time value, and returns `DefValue` if `S` does not contain a valid time indication.

Errors: None.

See also: StrToTime (1476), TryStrToTime (1487), StrToDateDef (1471)

30.12.222 strupper

Synopsis: Convert null-terminated string to all-uppercase

Declaration:
```
function strupper(p: pchar) : pchar
```

Visibility: default

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

For an example, see StrLower (1465)

Errors: None.

See also: StrLower (1465)

30.12.223 Supports

Synopsis: Check whether a class or given interface supports an interface

Declaration:
```
function Supports(const Instance: IInterface;const IID: TGUID;out Intf)
                  : Boolean;  Overload
function Supports(const Instance: TObject;const IID: TGUID;out Intf)
                  : Boolean;  Overload
function Supports(const Instance: IInterface;const IID: TGUID) : Boolean
                 ;  Overload
function Supports(const Instance: TObject;const IID: TGUID) : Boolean
                 ;  Overload
function Supports(const AClass: TClass;const IID: TGUID) : Boolean
                 ;  Overload
```

Visibility: default

Description: `Supports` checks whether `Instance` supports the interface identified by `IID`. It returns `True` if it is supported, `False`. Optionally, a pointer to the interface is returned to `Intf`.

Errors: None.

See also: StringToGUID (1461)

30.12.224 SysErrorMessage

Synopsis: Format a system error message.

Declaration: `function SysErrorMessage(ErrorCode: Integer) : String`

Visibility: default

Description: `SysErrorMessage` returns a string that describes the operating system error code `ErrorCode`.

Errors: This routine may not be implemented on all platforms.

See also: EOSError (1494)

30.12.225 SystemTimeToDateTime

Synopsis: Convert a system time to a `TDateTime` value.

Declaration: `function SystemTimeToDateTime(const SystemTime: TSystemTime) : TDateTime`

Visibility: default

Description: `SystemTimeToDateTime` converts a `TSystemTime` record to a `TDateTime` style date/time indication.

Errors: None.

See also: DateTimeToSystemTime (1396)

Listing: ./sysutex/ex22.pp

```
Program Example22;

{ This program demonstrates the SystemTimeToDateTime function }

Uses sysutils;

Var ST : TSystemTime;

Begin
  DateTimeToSystemTime(Now,ST);
  With St do
    begin
    Writeln ('Today is     ',year,'/',month,'/',Day);
    Writeln ('The time is ',Hour,':',minute,':',Second,'.',MilliSecond);
    end;
  Writeln ('Converted : ',DateTimeToStr(SystemTimeToDateTime(ST)));
End.
```

30.12.226 TextToFloat

Synopsis: Convert a buffer to a float value.

Declaration:
```
function TextToFloat(Buffer: PChar;var Value: Extended) : Boolean
function TextToFloat(Buffer: PChar;var Value: Extended;
                     const FormatSettings: TFormatSettings) : Boolean
function TextToFloat(Buffer: PChar;var Value;ValueType: TFloatValue)
                      : Boolean
function TextToFloat(Buffer: PChar;var Value;ValueType: TFloatValue;
                     const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TextToFloat` converts the string in `Buffer` to a floating point value. `Buffer` should contain a valid stroing representation of a floating point value (either in decimal or scientific notation). If the buffer contains a decimal value, then the decimal separator character can either be a '.' or the value of the `DecimalSeparator` variable.

The function returns `True` if the conversion was successful.

Errors: If there is an invalid character in the buffer, then the function returns `False`

See also: StrToFloat (1472), FloatToStr (1422), FormatFloat (1434)

Listing: ./sysutex/ex91.pp

```
Program Example91;

{ This program demonstrates the TextToFloat function }
{$mode objfpc}
{$h+ }

Uses SysUtils;

Const
  NrValues = 5;
  TestStr : Array[1..NrValues] of pchar =
            ('1,1','-0,2','1,2E-4','0','1E4');

Procedure Testit;

Var
  I : Integer;
  E : Extended;

begin
  Writeln('Using DecimalSeparator : ',DecimalSeparator);
  For I:=1 to NrValues do
    begin
    Writeln('Converting : ',TestStr[i]);
    If TextToFloat(TestStr[i],E) then
      Writeln('Converted value : ',E)
    else
      Writeln('Unable to convert value.');
    end;
end;

Begin
  DecimalSeparator:=',';
  Testit;
  DecimalSeparator:='.';
  Testit;
End.
```

30.12.227 Time

Synopsis: Returns the current time.

Declaration: `function Time : TDateTime`

Visibility: default

Description: `Time` returns the current time in `TDateTime` format. The date part of the `TDateTimeValue` is set to zero.

Errors: None.

See also: Now (1449), Date (1394)

Listing: ./sysutex/ex23.pp

```
Program Example23;

{ This program demonstrates the Time function }

Uses sysutils;

Begin
  WriteLn ('The time is : ',TimeToStr(Time));
End.
```

30.12.228 TimeStampToDateTime

Synopsis: Convert a TimeStamp value to a `TDateTime` value.

Declaration: `function TimeStampToDateTime(const TimeStamp: TTimeStamp) : TDateTime`

Visibility: default

Description: `TimeStampToDateTime` converts `TimeStamp` to a `TDateTime` format variable. It is the inverse operation of DateTimeToTimeStamp (1397).

Errors: None.

See also: DateTimeToTimeStamp (1397), TimeStampToMSecs (1481)

Listing: ./sysutex/ex24.pp

```
Program Example24;

{ This program demonstrates the TimeStampToDateTime function }

Uses sysutils;

Var TS : TTimeStamp;
    DT : TDateTime;

Begin
  TS:=DateTimeToTimeStamp (Now);
  With TS do
    begin
    WriteLn ('Now is ',time,' millisecond past midnight');
    WriteLn ('Today is ' ,Date,' days past 1/1/0001');
    end;
  DT:=TimeStampToDateTime(TS);
  WriteLn ('Together this is : ',DateTimeToStr(DT));
End.
```

30.12.229 TimeStampToMSecs

Synopsis: Converts a timestamp to a number of milliseconds.

Declaration:
```
function TimeStampToMSecs(const TimeStamp: TTimeStamp) : comp
```

Visibility: default

Description: `TimeStampToMSecs` converts `TimeStamp` to the number of seconds since `1/1/0001`.

Use `TTimeStamp` variables if you need to keep very precise track of time.

For an example, see MSecsToTimeStamp (1448).

Errors: None.

See also: MSecsToTimeStamp (1448), TimeStampToDateTime (1480)

30.12.230 TimeToStr

Synopsis: Convert a `TDateTime` time to a string using a predefined format.

Declaration:
```
function TimeToStr(Time: TDateTime) : String
```

Visibility: default

Description: `TimeToStr` converts the time in `Time` to a string. It uses the `ShortTimeFormat` variable to see what formatting needs to be applied. It is therefor entirely equivalent to a `FormatDateTime('t',Time)` call.

Errors: None.

Listing: ./sysutex/ex25.pp

```
Program Example25;

{ This program demonstrates the TimeToStr function }

Uses sysutils;

Begin
  WriteLn ('The current time is : ',TimeToStr(Time));
End.
```

30.12.231 Trim

Synopsis: Trim whitespace from the ends of a string.

Declaration:
```
function Trim(const S: String) : String
function Trim(const S: widestring) : widestring
```

Visibility: default

Description: `Trim` strips blank characters (spaces) at the beginning and end of `S` and returns the resulting string. Only `#32` characters are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: TrimLeft (1482), TrimRight (1483)

Listing: ./sysutex/ex84.pp

```
Program Example84;

{ This program demonstrates the Trim function }

Uses sysutils;
{$H+}

Procedure Testit (S : String);

begin
  WriteLn ('"',Trim(S),'"');
end;

Begin
  Testit ('  ha ha what gets lost ? ');
  Testit (#10#13'haha ');
  Testit ('               ');
End.
```

30.12.232 TrimLeft

Synopsis: Trim whitespace from the beginning of a string.

Declaration:
```
function TrimLeft(const S: String) : String
function TrimLeft(const S: widestring) : widestring
```

Visibility: default

Description: `TrimLeft` strips blank characters (spaces) at the beginning of `S` and returns the resulting string. Only `#32` characters are stripped. If the string contains only spaces, an empty string is returned.

Errors: None.

See also: Trim (1481), TrimRight (1483)

Listing: ./sysutex/ex85.pp

```
Program Example85;

{ This program demonstrates the TrimLeft function }

Uses sysutils;
{$H+}

Procedure Testit (S : String);

begin
  WriteLn ('"',TrimLeft(S),'"');
end;

Begin
  Testit ('  ha ha what gets lost ? ');
  Testit (#10#13'haha ');
  Testit ('               ');
End.
```

30.12.233 TrimRight

Synopsis: Trim whitespace from the end of a string.

Declaration:
```
function TrimRight(const S: String) : String
function TrimRight(const S: widestring) : widestring
```

Visibility: default

Description: `Trim` strips blank characters (spaces) at the end of `S` and returns the resulting string. Only `#32` characters are stripped. If the string contains only spaces, an empty string is returned.

Errors: None.

See also: Trim (1481), TrimLeft (1482)

Listing: ./sysutex/ex86.pp

```
Program Example86;

{ This program demonstrates the TrimRight function }

Uses sysutils;
{$H+}

Procedure Testit (S : String);

begin
  WriteIn ('"',TrimRight(S),'"');
end;

Begin
  Testit ('  ha ha what gets lost ? ');
  Testit (#10#13'haha ');
  Testit ('              ');
End.
```

30.12.234 TryEncodeDate

Synopsis: Try to encode a date, and indicate success.

Declaration:
```
function TryEncodeDate(Year: Word;Month: Word;Day: Word;
                       out Date: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDate` will check the validity of the `Year`, `Month` and `Day` arguments, and if they are all valid, then they will be encoded as a `TDateTime` value and returned in `D`. The function will return `True` in this case. If an inbvalid argument is passed, then `False` will be returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: EncodeDate (1403), DecodeDateFully (1399), DecodeDate (1398), TryEncodeTime (1484)

30.12.235 TryEncodeTime

Synopsis: Try to encode a time, and indicate success.

Declaration:
```
function TryEncodeTime(Hour: Word;Min: Word;Sec: Word;MSec: Word;
                       out Time: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeTime` will check the validity of the `Hour`, `Min`, `Sec` and `MSec` arguments, and will encode them in a `TDateTime` value which is returned in `T`. If the arguments are valid, then `True` is returned, otherwise `False` is returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: EncodeTime (1404), DecodeTime (1399), TryEncodeDate (1483)

30.12.236 TryFloatToCurr

Synopsis: Try to convert a float value to a currency value and report on success.

Declaration:
```
function TryFloatToCurr(const Value: Extended;var AResult: Currency)
                        : Boolean
```

Visibility: default

Description: `TryFloatToCurr` tries convert the `Value` floating point value to a `Currency` value. If successful, the function returns `True` and the resulting currency value is returned in `AResult`. It checks whether `Value` is in the valid range of currencies (determined by MinCurrency (1358) and MaxCurrency (1357)). If not, `False` is returned.

Errors: If `Value` is out of range, `False` is returned.

See also: FloatToCurr (1421), MinCurrency (1358), MaxCurrency (1357)

30.12.237 TryStrToBool

Synopsis: Try to convert a string to a boolean value

Declaration:
```
function TryStrToBool(const S: String;out Value: Boolean) : Boolean
```

Visibility: default

Description: `TryStrToBool` tries to convert the string `S` to a boolean value, and returns this value in `Value`. In this case, the function returns `True`. If `S` does not contain a valid boolean string, the function returns `False`, and the contents of `Value` is indetermined.

Valid boolean string constants are in the FalseBoolStrs (1368) (for `False` values) and TrueBoolStrs (1370) (for `True` values) variables.

See also: StrToBool (1469), StrToBoolDef (1469)

30.12.238 TryStrToCurr

Synopsis: Try to convert a string to a currency

Declaration:
```
function TryStrToCurr(const S: String;var Value: Currency) : Boolean
```

Visibility: default

Description: `TryStrToCurr` converts the string `S` to a currency value and returns the value in `Value`. The function returns `True` if it was successful, `False` if not. This is contrary to StrToCurr (1470), which raises an exception when the conversion fails.

The function takes into account locale information.

See also: StrToCurr (1470), TextToFloat (1478)

30.12.239 TryStrToDate

Synopsis: Try to convert a string with a date indication to a TDateTime value

Declaration:
```
function TryStrToDate(const S: String;out Value: TDateTime) : Boolean
```

Visibility: default

Description: `TryStrToDate` tries to convert the string `S` to a `TDateTime` date value, and stores the date in `Value`. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

The function returns `True` if the string contained a valid date indication, `False` otherwise.

See also: StrToDate (1470), StrToTime (1476), TryStrToTime (1487), TryStrToDateTime (1485), DateToStr (1397), TimeToStr (1481)

30.12.240 TryStrToDateTime

Synopsis: Try to convert a string with date/time indication to a TDateTime value

Declaration:
```
function TryStrToDateTime(const S: String;out Value: TDateTime)
                                : Boolean
```

Visibility: default

Description: `TryStrToDateTime` tries to convert the string `S` to a `TDateTime` date and time value, and stores the result in `Value`. The date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month (This is *not* supported in Delphi). The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

The function returns `True` if the string contained a valid date and time indication, `False` otherwise.

See also: TryStrToDate (1485), TryStrToTime (1487), StrToDateTime (1471), StrToTime (1476), DateToStr (1397), TimeToStr (1481)

30.12.241 TryStrToFloat

Synopsis: Try to convert a string to a float.

Declaration:
```
function TryStrToFloat(const S: String;var Value: Single) : Boolean
function TryStrToFloat(const S: String;var Value: Single;
                       const FormatSettings: TFormatSettings) : Boolean
function TryStrToFloat(const S: String;var Value: Double) : Boolean
function TryStrToFloat(const S: String;var Value: Double;
                       const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToFloat` tries to convert the string `S` to a floating point value, and stores the result in `Value`. It returns `True` if the operation was succesful, and `False` if it failed. This operation takes into account the system settings for floating point representations.

Errors: On error, `False` is returned.

See also: StrToFloat (1472)

30.12.242 TryStrToInt

Synopsis: Try to convert a string to an integer, and report on success.

Declaration: `function TryStrToInt(const s: String;var i: Integer) : Boolean`

Visibility: default

Description: `TryStrToInt` tries to convert the string `S` to an integer, and returns `True` if this was succesful. In that case the converted integer is returned in `I`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: StrToInt (1474), TryStrToInt64 (1486), StrToIntDef (1475), StrToInt64 (1474), StrToInt64Def (1475)

30.12.243 TryStrToInt64

Synopsis: Try to convert a string to an int64 value, and report on success.

Declaration: `function TryStrToInt64(const s: String;var i: Int64) : Boolean`

Visibility: default

Description: `TryStrToInt64` tries to convert the string `S` to a Int64 value, and returns this value in `I` if successful. If the conversion was succesful, the function result is `True`, or `False` otherwise. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the $ character) are supported.

Errors: None. On error, `False` is returned.

See also: StrToInt64 (1474), StrToInt64Def (1475), StrToInt (1474), TryStrToInt (1486), StrToIntDef (1475)

30.12.244 TryStrToQWord

Synopsis: Try to convert a string to a QWord value, and report on success

Declaration: `function TryStrToQWord(const s: String;var Q: QWord) : Boolean`

Visibility: default

Description: `TryStrToQWord` tries to convert the string `S` to a valid QWord (unsigned 64-bit) value, and stores the result in `I`. If the conversion fails, the function returns `False`, else it returns `True`.

See also: StrToQWord (1476), StrToQWordDef (1476), TryStrToInt64 (1486), TryStrToInt (1486)

30.12.245 TryStrToTime

Synopsis: Try to convert a string with a time indication to a TDateTime value

Declaration: `function TryStrToTime(const S: String;out Value: TDateTime) : Boolean`

Visibility: default

Description: `TryStrToTime` tries to convert the string `S` to a `TDateTime` time value, and stores the result in `Value`. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

The function returns `True` if the string contained a valid time indication, `False` otherwise.

See also: TryStrToDate (1485), TryStrToDateTime (1485), StrToDate (1470), StrToTime (1476), DateToStr (1397), TimeToStr (1481)

30.12.246 UnhookSignal

Synopsis: UnHook a specified signal

Declaration: `procedure UnhookSignal(RtlSigNum: Integer;OnlyIfHooked: Boolean)`

Visibility: default

Description: This function is declared for Kylix compatibility, but is not implemented.

30.12.247 UpperCase

Synopsis: Return an uppercase version of a string.

Declaration: `function UpperCase(const s: String) : String`

Visibility: default

Description: `UpperCase` returns the uppercase equivalent of `S`. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the `UpCase` function of the system unit, and is provided for compatiibility only.

Errors: None.

See also: AnsiLowerCase (1376), LowerCase (1448), AnsiUpperCase (1384)

Listing: ./sysutex/ex87.pp

```
Program Example87;

{ This program demonstrates the UpperCase function }

Uses sysutils;

Begin
  WriteLn (UpperCase('this will come OUT ALL uPpErCaSe !'));
End.
```

30.12.248 VendorName

Synopsis: Returns the application vendor name.

Declaration:
```
function VendorName : String
```

Visibility: default

Description: `VendorName` returns the application vendor name. This function does not do anything by itself, but uses the OnGetVendorName (1369) callback to get the application vendor name.

Errors: If OnGetVendorName (1369) is not set, an empty string is returned.

See also: OnGetVendorName (1369), TGetVendorNameEvent (1365)

30.12.249 WideCompareStr

Synopsis: Compare two widestrings (case sensitive)

Declaration:
```
function WideCompareStr(const s1: WideString;const s2: WideString)
                                : PtrInt
```

Visibility: default

Description: `WideCompareStr` compares two widestrings and returns the following result:

< 0if `S1<S2`.

0if `S1=S2`.

> 0if `S1>S2`.

The comparision takes into account wide characters, i.e. it takes care of strange accented characters. Contrary to WideCompareText (1488), the comparision is case sensitive.

Errors: None.

See also: WideCompareText (1488), WideSameStr (1490), WideSameText (1490)

30.12.250 WideCompareText

Synopsis: Compare two widestrings (ignoring case).

Declaration:
```
function WideCompareText(const s1: WideString;const s2: WideString)
                                : PtrInt
```

Visibility: default

Description: `WideCompareStr` compares two widestrings and returns the following result:

< 0if `S1<S2`.

0if `S1=S2`.

> 0if `S1>S2`.

The comparision takes into account wide characters, i.e. it takes care of strange accented characters. Contrary to WideCompareStr (1488), the comparision is case insensitive.

Errors: None.

See also: WideCompareStr (1488), WideSameStr (1490), WideSameText (1490)

30.12.251 WideFmtStr

Synopsis: Widestring format

Declaration:
```
procedure WideFmtStr(var Res: WideString;const Fmt: WideString;
                     const args: Array of const)
procedure WideFmtStr(var Res: WideString;const Fmt: WideString;
                     const args: Array of const;
                     const FormatSettings: TFormatSettings)
```

Visibility: default

Description: `WideFmtStr` formats `Args` according to the format string in `Fmt` and returns the resulting string in `Res`.

See also: WideFormat (1489), WideFormatBuf (1489), Format (1427)

30.12.252 WideFormat

Synopsis: Format a wide string.

Declaration:
```
function WideFormat(const Fmt: WideString;const Args: Array of const)
                    : WideString
function WideFormat(const Fmt: WideString;const Args: Array of const;
                    const FormatSettings: TFormatSettings) : WideString
```

Visibility: default

Description: `WideFormat` does the same as Format (1427) but accepts as a formatting string a WideString. The resulting string is also a WideString.

For more information about the used formatting characters, see the Format (1427) string.

See also: Format (1427)

30.12.253 WideFormatBuf

Synopsis: Format widestring in a buffer.

Declaration:
```
function WideFormatBuf(var Buffer;BufLen: Cardinal;const Fmt;
                       fmtLen: Cardinal;const Args: Array of const)
                        : Cardinal
function WideFormatBuf(var Buffer;BufLen: Cardinal;const Fmt;
                       fmtLen: Cardinal;const Args: Array of const;
                       const FormatSettings: TFormatSettings) : Cardinal
```

Visibility: default

Description: `WideFormatBuf` calls simply WideFormat (1489) with `Fmt` (with length `FmtLen` bytes) and stores maximum `BufLen` bytes in the buffer `buf`. It returns the number of copied bytes.

See also: WideFmtStr (1489), WideFormat (1489), Format (1427), FormatBuf (1432)

30.12.254 WideLowerCase

Synopsis: Change a widestring to all-lowercase.

Declaration:
```
function WideLowerCase(const s: WideString) : WideString
```

Visibility: default

Description: `WideLowerCase` converts the string `S` to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so spcial characters are converted correctly as well.

Remark: On Unix-like platforms, a widestring manager must be installed for this function to work correctly.

Errors: None.

See also: WideUpperCase (1490)

30.12.255 WideSameStr

Synopsis: Check whether two widestrings are the same (case sensitive)

Declaration:
```
function WideSameStr(const s1: WideString;const s2: WideString)
                           : Boolean
```

Visibility: default

Description: `WideSameStr` returns `True` if WideCompareStr (1488) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: WideSameText (1490), WideCompareStr (1488), WideCompareText (1488), AnsiSameStr (1378)

30.12.256 WideSameText

Synopsis: Check whether two widestrings are the same (ignoring case)

Declaration:
```
function WideSameText(const s1: WideString;const s2: WideString)
                            : Boolean
```

Visibility: default

Description: `WideSameText` returns `True` if WideCompareText (1488) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: WideSameStr (1490), WideCompareStr (1488), WideCompareText (1488), AnsiSameText (1378)

30.12.257 WideUpperCase

Synopsis: Change a widestring to all-lowercase.

Declaration:
```
function WideUpperCase(const s: WideString) : WideString
```

Visibility: default

Description: `WideUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so spcial characters are converted correctly as well.

Remark: On Unix-like platforms, a widestring manager must be installed for this function to work correctly.

Errors: None.

See also: WideLowerCase (1490)

30.12.258 WrapText

Synopsis: Word-wrap a text.

Declaration:
```
function WrapText(const Line: String;const BreakStr: String;
                  const BreakChars: TSysCharSet;MaxCol: Integer) : String
function WrapText(const Line: String;MaxCol: Integer) : String
```

Visibility: default

Description: `WrapText` does a wordwrap at column `MaxCol` of the string in `Line`. It breaks the string only at characters which are in `BreakChars` (default whitespace and hyphen) and inserts then the string `BreakStr` (default the lineending character for the current OS).

See also: StringReplace (1461)

30.13 EAbort

30.13.1 Description

`Abort` is raised by the Abort (1371) procedure. It is not displayed in GUI applications, and serves only to immediatly abort the current procedure, and return control to the main program loop.

30.14 EAbstractError

30.14.1 Description

`EAbstractError` is raised when an abstract error occurs, i.e. when an unimplemented abstract method is called.

30.15 EAccessViolation

30.15.1 Description

`EAccessViolation` is raised when the OS reports an Access Violation, i.e. when invalid memory is accessed.

30.16 EAssertionFailed

30.16.1 Description

`EAssertionFailed` is raised when an application that is compiled with assertions, encounters an invalid assertion.

30.17 EBusError

30.17.1 Description

`EBusError` is raised in case of a bus error.

30.18 EControlC

30.18.1 Description

EControlC is raised when the user has pressed CTRL-C in a console application.

30.19 EConvertError

30.19.1 Description

EConvertError is raised by the various conversion routines in the SysUtils unit. The message will contain more specific error information.

30.20 EDivByZero

30.20.1 Description

EDivByZero is used when the operating system or CPU signals a division by zero error.

30.21 EExternal

30.21.1 Description

EExternal is the base exception for all external exceptions, as reported by the CPU or operating system, as opposed to internal exceptions, which are raised by the program itself. The SysUtils unit converts all operating system errors to descendents of EExternal.

30.22 EExternalException

30.22.1 Description

EExternalException is raised when an external routine raises an exception.

30.23 EFormatError

30.23.1 Description

EformatError is raised in case of an error in one of the various Format (1427) functions.

30.24 EHeapMemoryError

30.24.1 Description

EHeapMemoryError is raised when an error occurs in heap (dynamically allocated) memory.

30.25 EInOutError

30.25.1 Description

`EInOutError` is raised when a IO routine of Free Pascal returns an error. The error is converted to an `EInOutError` only if the input/output checking feature of FPC is turned on. The error code of the input/output operation is returned in ErrorCode (**??**).

30.26 EInterror

30.26.1 Description

`EInterror` is used when the operating system or CPU signals an integer operation error, e.g., an overflow.

30.27 EIntfCastError

30.27.1 Description

`EIntfCastError` is raised when an invalid interface cast is encountered.

30.28 EIntOverflow

30.28.1 Description

`EIntOverflow` is used when the operating system or CPU signals a integer overflow error.

30.29 EInvalidCast

30.29.1 Description

`EInvalidCast` is raised when an invalid typecast error (using the `as` operator) is encountered.

30.30 EInvalidContainer

30.30.1 Description

`EInvalidContainer` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.31 EInvalidInsert

30.31.1 Description

`EInvalidInsert` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.32 EInvalidOp

30.32.1 Description

`EInvalidOp` is raised when an invalid operation is encountered.

30.33 EInvalidPointer

30.33.1 Description

`EInvalidPointer` is raised when an invalid heap pointer is used.

30.34 EMathError

30.34.1 Description

`EMathError` is used when the operating system or CPU signals a floating point overflow error.

30.35 ENoThreadSupport

30.35.1 Description

`ENoThreadSupport` is raised when some thread routines are invoked, and thread support was not enabled when the program was compiled.

30.36 ENoWideStringSupport

30.36.1 Description

`ENoWideStringSupport` is the exception raised when a run-time 233 occurs, i.e. when widestring routines are called and the application does not contain widestring support.

30.37 EOSError

30.37.1 Description

`EOSError` is raised when some Operating System call fails. The ErrorCode (**??**) property contains the operating system error code.

30.38 EOutOfMemory

30.38.1 Description

`EOutOfMemory` occurs when memory can no longer be allocated on the heap. An instance of `EOutOfMemory` is allocated on the heap at program startup, so it is available when needed.

30.39 EOverflow

30.39.1 Description

`EOverflow` occurs when a float operation overflows. (i.e. result is too big to represent).

30.40 EPackageError

30.40.1 Description

`EPackageError` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.41 EPrivilege

30.41.1 Description

`EPrivilege` is raised when the OS reports that an invalid instruction was executed.

30.42 EPropReadOnly

30.42.1 Description

`EPropReadOnly` is raised when an attempt is made to write to a read-only property.

30.43 EPropWriteOnly

30.43.1 Description

`EPropWriteOnly` is raised when an attempt is made to read from a write-only property.

30.44 ERangeError

30.44.1 Description

`ERangeError` is raised by the Free Pascal runtime library if range checking is on, and a range check error occurs.

30.45 ESafecallException

30.45.1 Description

`ESafecallException` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.46 EStackOverflow

30.46.1 Description

`EStackOverflow` occurs when the stack has grown too big (e.g. by infinite recursion).

30.47 EUnderflow

30.47.1 Description

`EOverflow` occurs when a float operation underflows (i.e. result is too small to represent).

30.48 EVariantError

30.48.1 Description

`EVariantError` is raised by the internal variant routines.

30.48.2 Method overview

Page	Property	Description
1496	CreateCode	Create an instance of `EVariantError` with a particular error code.

30.48.3 EVariantError.CreateCode

Synopsis: Create an instance of `EVariantError` with a particular error code.

Declaration: `constructor CreateCode(Code: LongInt)`

Visibility: default

Description: `CreateCode` calls the inherited constructor, and sets the ErrCode (**??**) property to `Code`.

See also: EVariantError.ErrCode (**??**)

30.49 Exception

30.49.1 Description

`Exception` is the base class for all exception handling routines in the RTL and FCL. While it is possible to raise an exception with any class descending from `TObject`, it is recommended to use `Exception` as the basis of exception class objects: the `Exception` class introduces properties to associate a message and a help context with the exception being raised. What is more, the SysUtils unit sets the necessary hooks to catch and display unhandled exceptions: in such cases, the message displayed to the end user, will be the message stored in the exception class.

30.49.2 Method overview

Page	Property	Description
1497	Create	Constructs a new exception object with a given message.
1497	CreateFmt	Constructs a new exception object and formats a new message.
1498	CreateFmtHelp	Constructs a new exception object and sets the help context and formats the message
1498	CreateHelp	Constructs a new exception object and sets the help context.
1497	CreateRes	Constructs a new exception object and gets the message from a resource.
1498	CreateResFmt	Constructs a new exception object and formats the message from a reasource.
1499	CreateResFmtHelp	Constructs a new exception object and sets the help context and formats the message from a resource
1498	CreateResHelp	Constructs a new exception object and sets the help context and gets the message from a resource

30.49.3 Property overview

Page	Property	Access	Description
1499	HelpContext	rw	Help context associated with the exception.
1499	Message	rw	Message associated with the exception.

30.49.4 Exception.Create

Synopsis: Constructs a new exception object with a given message.

Declaration:
```
constructor Create(const msg: String)
```

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.CreateFmt (1497), Exception.Message (1499)

30.49.5 Exception.CreateFmt

Synopsis: Constructs a new exception object and formats a new message.

Declaration:
```
constructor CreateFmt(const msg: String;const args: Array of const)
```

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.Create (1497), Exception.Message (1499), Format (1427)

30.49.6 Exception.CreateRes

Synopsis: Constructs a new exception object and gets the message from a resource.

Declaration:
```
constructor CreateRes(ResString: PString)
```

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.Create (1497), Exception.CreateFmt (1497), Exception.CreateResFmt (1498), Exception.Message (1499)

30.49.7 Exception.CreateResFmt

Synopsis: Constructs a new exception object and formats the message from a reasource.

Declaration:
```
constructor CreateResFmt(ResString: PString;const Args: Array of const)
```

Visibility: public

Description: `CreateResFmt` does the same as CreateFmt (1497), but fetches the message from the resource string `ResString`.

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.Create (1497), Exception.CreateFmt (1497), Exception.CreateRes (1497), Exception.Message (1499)

30.49.8 Exception.CreateHelp

Synopsis: Constructs a new exception object and sets the help context.

Declaration:
```
constructor CreateHelp(const Msg: String;AHelpContext: Integer)
```

Visibility: public

Description: `CreateHelp` does the same as the Create (1497) constructor, but additionally stores `AHelpContext` in the HelpContext (1499) property.

See also: Exception.Create (1497)

30.49.9 Exception.CreateFmtHelp

Synopsis: Constructs a new exception object and sets the help context and formats the message

Declaration:
```
constructor CreateFmtHelp(const Msg: String;const Args: Array of const;
                          AHelpContext: Integer)
```

Visibility: public

Description: `CreateFmtHelp` does the same as the CreateFmt (1497) constructor, but additionally stores `AHelpContext` in the HelpContext (1499) property.

See also: Exception.CreateFmt (1497)

30.49.10 Exception.CreateResHelp

Synopsis: Constructs a new exception object and sets the help context and gets the message from a resource

Declaration:
```
constructor CreateResHelp(ResString: PString;AHelpContext: Integer)
```

Visibility: public

Description: `CreateResHelp` does the same as the CreateRes (1497) constructor, but additionally stores `AHelpContext` in the HelpContext (1499) property.

See also: Exception.CreateRes (1497)

30.49.11 Exception.CreateResFmtHelp

Synopsis: Constructs a new exception object and sets the help context and formats the message from a resource

Declaration:
```
constructor CreateResFmtHelp(ResString: PString;
                             const Args: Array of const;
                             AHelpContext: Integer)
```

Visibility: public

Description: `CreateResFmtHelp` does the same as the CreateResFmt (1498) constructor, but additionally stores `AHelpContext` in the HelpContext (1499) property.

See also: Exception.CreateResFmt (1498)

30.49.12 Exception.HelpContext

Synopsis: Help context associated with the exception.

Declaration:
```
Property HelpContext : LongInt
```

Visibility: public

Access: Read,Write

Description: `HelpContext` is the help context associated with the exception, and can be used to provide context-sensitive help when the exception error message is displayed. It should be set in the exception constructor.

See also: Exception.CreateHelp (1498), Exception.Message (1499)

30.49.13 Exception.Message

Synopsis: Message associated with the exception.

Declaration:
```
Property Message : String
```

Visibility: public

Access: Read,Write

Description: `Message` provides additional information about the exception. It is shown to the user in e.g. the ShowException (1454) routine, and should be set in the constructor when the exception is raised.

See also: Exception.Create (1497), Exception.HelpContext (1499)

30.50 EZeroDivide

30.50.1 Description

`EZeroDivide` occurs when a float division by zero occurs.

30.51 IReadWriteSync

30.51.1 Description

`IReadWriteSync` is an interface for synchronizing read/write operations. Writers are always guaranteed to have exclusive access: readers may or may not have simultaneous access, depending on the implementation.

30.51.2 Method overview

Page	Property	Description
1500	BeginRead	Start a read operation.
1500	BeginWrite	Start a write operation.
1500	EndRead	End a read operation
1501	EndWrite	End a write operation.

30.51.3 IReadWriteSync.BeginRead

Synopsis: Start a read operation.

Declaration: `procedure BeginRead`

Visibility: default

Description: `BeginRead` indicates that a read operation is about to be started. If a write operation is in progress, then the call will block untill the write operation finished. Depending on the implementation the call may also block if another read operation is in progress.

After `BeginRead`, any write operation started with BeginWrite (1500) will block until EndRead (1500) is called.

See also: IReadWriteSync.EndRead (1500), IReadWriteSync.BeginWrite (1500), IReadWriteSync.EndWrite (1501)

30.51.4 IReadWriteSync.EndRead

Synopsis: End a read operation

Declaration: `procedure EndRead`

Visibility: default

Description: `EndRead` signals the end of a read operation. If there was any blocked write operation, that will be unblocked by a call to `EndRead`.

See also: IReadWriteSync.BeginRead (1500), IReadWriteSync.BeginWrite (1500), IReadWriteSync.EndWrite (1501)

30.51.5 IReadWriteSync.BeginWrite

Synopsis: Start a write operation.

Declaration: `function BeginWrite : Boolean`

Visibility: default

Description: `BeginWrite` signals the begin of a write operation. This call will block if any other read or write operation is currently in progress. It will resumt only after all other read or write operations have finished.

See also: IReadWriteSync.EndRead (1500), IReadWriteSync.EndWrite (1501), IReadWriteSync.BeginRead (1500)

30.51.6 IReadWriteSync.EndWrite

Synopsis: End a write operation.

Declaration: `procedure EndWrite`

Visibility: default

Description: `EndWrite` signals the end of a write operation. After the call to `EndWrite` any other read or write operations can start.

See also: IReadWriteSync.EndRead (1500), IReadWriteSync.EndWrite (1501), IReadWriteSync.BeginRead (1500)

30.52 TMultiReadExclusiveWriteSynchronizer

30.52.1 Description

`TMultiReadExclusiveWriteSynchronizer` is a default implementation of the IReadWriteSync (1500) interface. It uses a single mutex to protect access to the read/write resource, resulting in a single thread having access to the resource.

30.52.2 Method overview

Page	Property	Description
1502	Beginread	Request read access to the resource
1502	Beginwrite	Request write access to the resource.
1501	Create	Create a new instance of the `TMultiReadExclusiveWriteSynchronizer` class
1502	Destroy	Destroys the `TMultiReadExclusiveWriteSynchronizer` instance
1503	Endread	Release read access to the resource
1502	Endwrite	Release write access to the resource

30.52.3 TMultiReadExclusiveWriteSynchronizer.Create

Synopsis: Create a new instance of the `TMultiReadExclusiveWriteSynchronizer` class

Declaration: `constructor Create; Virtual`

Visibility: public

Description: `Create` creates a new instance of `TMultiReadExclusiveWriteSynchronizer`. It initializes a `TRTLCriticalSection`.

Errors: None.

See also: #rtl.system.TRTLCriticalSection (1195)

30.52.4 TMultiReadExclusiveWriteSynchronizer.Destroy

Synopsis: Destroys the `TMultiReadExclusiveWriteSynchronizer` instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Create` destroys the instance of `TMultiReadExclusiveWriteSynchronizer`. It frees the `TRTLCriticalSection` it initialized, and calls the inherited destructor.

Errors: None.

See also: #rtl.system.TRTLCriticalSection (1195)

30.52.5 TMultiReadExclusiveWriteSynchronizer.Beginwrite

Synopsis: Request write access to the resource.

Declaration: `function Beginwrite : Boolean`

Visibility: public

Description: `Beginwrite` is the implementation of `IReadWriteSync.BeginWrite`. It simply enters the critical section, and returns `True`.

Errors: None.

See also: IReadWriteSync.BeginWrite (1500), TMultiReadExclusiveWriteSynchronizer.EndWrite (1502)

30.52.6 TMultiReadExclusiveWriteSynchronizer.Endwrite

Synopsis: Release write access to the resource

Declaration: `procedure Endwrite`

Visibility: public

Description: `Beginwrite` is the implementation of `IReadWriteSync.EndWrite`. It simply leaves the critical section.

Errors: None.

See also: IReadWriteSync.EndWrite (1501), TMultiReadExclusiveWriteSynchronizer.BeginWrite (1502)

30.52.7 TMultiReadExclusiveWriteSynchronizer.Beginread

Synopsis: Request read access to the resource

Declaration: `procedure Beginread`

Visibility: public

Description: `BeginRead` is the implementation of `IReadWriteSync.BeginRead`. It simply attempts to enter the critical section.

Errors: None.

See also: IReadWriteSync.BeginRead (1500), TMultiReadExclusiveWriteSynchronizer.EndRead (1503)

30.52.8 TMultiReadExclusiveWriteSynchronizer.Endread

Synopsis: Release read access to the resource

Declaration: `procedure Endread`

Visibility: public

Description: `EndRead` is the implementation of `IReadWriteSync.EndRead`. It simply leaves the critical section.

Errors: None.

See also: IReadWriteSync.EndRead (1500), TMultiReadExclusiveWriteSynchronizer.BeginRead (1502)

Chapter 31

Reference for unit 'typinfo'

31.1 Auxiliary functions

Other typinfo related functions.

Table 31.1:

Name	Description
GetEnumName (1512)	Get an enumerated type element name
GetEnumValue (1514)	Get ordinal number of an enumerated type, based on the name.
GetEnumNameCount (1513)	Get number of elements in an enumerated type.
GetTypeData (1526)	Skip type name and return a pointer to the type data
SetToString (1535)	Convert a set to its string representation
StringToSet (1536)	Convert a string representation of a set to a set

31.2 Getting or setting property values

Functions to set or set a property's value.

31.3 Examining published property information

Functions for retrieving or examining property information

31.4 Used units

31.5 Overview

The TypeInfo unit contains many routines which can be used for the querying of the Run-Time Type Information (RTTI) which is generated by the compiler for classes that are compiled under the `{$M+}` switch. This information can be used to retrieve or set property values for published properties for totally unknown classes. In particular, it can be used to stream classes. The `TPersistent`

Table 31.2:

Name	Description
GetEnumProp (1513)	Return the value of an enumerated type property
GetFloatProp (1515)	Return the value of a float property
GetInt64Prop (1516)	Return the value of an Int64 property
GetMethodProp (1517)	Return the value of a procedural type property
GetObjectProp (1519)	Return the value of an object property
GetOrdProp (1520)	Return the value of an ordinal type property
GetPropValue (1524)	Return the value of a property as a variant
GetSetProp (1524)	Return the value of a set property
GetStrProp (1525)	Return the value of a string property
GetWideStrProp (1527)	Return the value of a widestring property
GetVariantProp (1527)	Return the value of a variant property
SetEnumProp (1531)	Set the value of an enumerated type property
SetFloatProp (1531)	Set the value of a float property
SetInt64Prop (1531)	Set the value of an Int64 property
SetMethodProp (1532)	Set the value of a procedural type property
SetObjectProp (1533)	Set the value of an object property
SetOrdProp (1533)	Set the value of an ordinal type property
SetPropValue (1534)	Set the value of a property trhough a variant
SetSetProp (1534)	Set the value of a set property
SetStrProp (1534)	Set the value of a string property
SetWideStrProp (1536)	Set the value of a widestring property
SetVariantProp (1536)	Set the value of a variant property

class in the Classes unit is compiled in the `{$M+}` state and serves as the base class for all classes that need to be streamed.

The unit should be compatible to the Delphi 5 unit with the same name. The only calls that are still missing are the Variant calls, since Free Pascal does not support the variant type yet.

The examples in this chapter use a rttiobj file, which contains an object that has a published property of all supported types. It also contains some auxiliary routines and definitions.

31.6 Constants, types and variables

31.6.1 Constants

```
BooleanIdents : Array[Boolean] of String = ('False','True' )
```

Names for boolean values

```
DotSep : String = '.'
```

Name separator character

```
OnGetPropValue : TGetPropValue = nil
```

This callback is set by the variants unit to enable reading of properties as a variant. If set, it is called by the GetPropValue (1524) function.

Table 31.3:

Name	Description
FindPropInfo (1511)	Getting property type information, With error checking.
GetPropInfo (1521)	Getting property type information, No error checking.
GetPropInfos (1522)	Find property information of a certain kind
GetObjectPropClass (1520)	Return the declared class of an object property
GetPropList (1523)	Get a list of all published properties
IsPublishedProp (1527)	Is a property published
IsStoredProp (1528)	Is a property stored
PropIsType (1529)	Is a property of a certain kind
PropType (1530)	Return the type of a property

Table 31.4: Used units by unit 'typinfo'

Name	Page
sysutils	1350

```
OnGetVariantprop : TGetVariantProp = nil
```

This callback is set by the variants unit to enable reading of variant properties If set, it is called by the GetVariantProp (1527) function.

```
OnSetPropValue : TSetPropValue = nil
```

This callback is set by the variants unit to enable writing of properties as a variant. If set, it is called by the SetPropValue (1534) function.

```
OnSetVariantprop : TSetVariantProp = nil
```

This callback is set by the variants unit to enable writing of variant properties. If set, it is called by the GetVariantProp (1527) function.

```
ptConst = 3
```

Constant used in acces method

```
ptField = 0
```

Property acces directly from field

```
ptStatic = 1
```

Property acces via static method

```
ptVirtual = 2
```

Property acces via virtual method

```
tkAny = [Low ( TTypeKind ) ..High ( TTypeKind ) ]
```

Any property type

```
tkMethods = [tkMethod]
```

Only method properties. (event handlers)

```
tkProperties = tkAny - tkMethods - [tkUnknown]
```

Real properties. (not methods)

```
tkString = tkSString
```

Alias for the `tsSString` enumeration value

31.6.2 Types

```
PPropInfo = ^TPropInfo
```

Pointer to TPropInfo (1510) record

```
PPropList = ^TPropList
```

Pointer to TPropList (1510)

```
PPTypeInfo = ^PTypeInfo
```

Pointer to PTypeInfo (1507) pointer

```
PTypeData = ^TTypeData
```

Pointer to TTypeData (1510) record.

```
PTypeInfo = ^TTypeInfo
```

Pointer to TTypeInfo (1510) record

```
ShortStringBase =
```

`ShortStringBase` is the base definition of a short string.

```
TFloatType = (ftSingle,ftDouble,ftExtended,ftComp,ftCurr)
```

The size of a float type.

```
TGetPropValue = function(Instance: TObject;const PropName: String;
                         PreferStrings: Boolean) : Variant
```

The callback function must return the property with name `PropName` of instance `Instance`. If `PreferStrings` is true, it should favour converting the property to a string value. The function needs to return the variant with the property value.

Table 31.5: Enumeration values for type TFloatType

Value	Explanation
ftComp	Comp-type float
ftCurr	Currency-type float
ftDouble	Double-sized float
ftExtended	Extended-size float
ftSingle	Single-sized float

```
TGetVariantProp = function(Instance: TObject;PropInfo: PPropInfo)
                           : Variant
```

The callback function must return the variant property with name `PropName` of instance `Instance`.

```
TIntfFlag = (ifHasGuid,ifDispInterface,ifDispatch,ifHasStrGUID)
```

Table 31.6: Enumeration values for type TIntfFlag

Value	Explanation
ifDispatch	Interface is a dispatch interface
ifDispInterface	Interface is a dual dispatch interface
ifHasGuid	Interface has GUID identifier
ifHasStrGUID	Interface has a string GUID identifier

Type of interface.

```
TIntfFlags= Set of (ifDispatch,ifDispInterface,ifHasGuid,ifHasStrGUID)
```

Set of TIntfFlag (1508).

```
TIntfFlagsBase= Set of (ifDispatch,ifDispInterface,ifHasGuid,
                        ifHasStrGUID)
```

Set of TIntfFlag (1508).

```
TMethodKind = (mkProcedure,mkFunction,mkConstructor,mkDestructor,
               mkClassProcedure,mkClassFunction)
```

Method type description

```
TOrdType = (otSByte,otUByte,otSWord,otUWord,otSLong,otULong)
```

If the property is and ordinal type, then `TOrdType` determines the size and sign of the ordinal type:

```
TParamFlag = (pfVar,pfConst,pfArray,pfAddress,pfReference,pfOut)
```

`TParamFlag` describes a parameter.

Table 31.7: Enumeration values for type TMethodKind

Value	Explanation
mkClassFunction	Class function
mkClassProcedure	Class procedure
mkConstructor	Class constructor
mkDestructor	Class Desctructor
mkFunction	Function method
mkProcedure	Procedure method.

Table 31.8: Enumeration values for type TOrdType

Value	Explanation
otSByte	Signed byte
otSLong	Signed longint
otSWord	Signed word
otUByte	Unsigned byte
otULong	Unsigned longing (Cardinal)
otUWord	Unsigned word

```
TParamFlags= Set of (pfAddress,pfArray,pfConst,pfOut,pfReference,pfVar)
```

The kind of parameter for a method

```
TProcInfoProc = procedure(PropInfo: PPropInfo) of object
```

Property info callback method

```
TPropData = packed record
  PropCount : Word;
  PropList : record
    _alignmentdummy : ptrint;
  end;
end
```

The `TPropData` record is not used, but is provided for completeness and compatibility with Delphi.

```
TPropInfo = packed record
  PropType : PTypeInfo;
  GetProc : Pointer;
  SetProc : Pointer;
  StoredProc : Pointer;
  Index : Integer;
  Default : LongInt;
  NameIndex : SmallInt;
  PropProcs : Byte;
  Name : ShortString;
end
```

Table 31.9: Enumeration values for type TParamFlag

Value	Explanation
pfAddress	Parameter is passed by address
pfArray	Parameter is an array parameter
pfConst	Parameter is a const parameter (i.e. cannot be modified)
pfOut	Parameter is a string parameter
pfReference	Parameter is passed by reference
pfVar	Parameter is a var parameter (passed by reference)

The `TPropInfo` record describes one published property of a class. The property information of a class are stored as an array of `TPropInfo` records.

The `Name` field is stored not with 255 characters, but with just as many characters as required to store the name.

```
TPropList = Array[0..65535] of PPropInfo
```

Array of property information pointers

```
TSetPropValue = procedure(Instance: TObject;const PropName: String;
                          const Value: Variant)
```

The callback function must set the property with name `PropName` of instance `Instance` to `Value`.

```
TSetVariantProp = procedure(Instance: TObject;PropInfo: PPropInfo;
                            const Value: Variant)
```

The callback function must set the variant property with name `PropName` of instance to `Value`.

```
TTypeData = packed record
end
```

If the typeinfo kind is `tkClass`, then the property information follows the `UnitName` string, as an array of TPropInfo (1510) records.

```
TTypeInfo = record
  Kind : TTypeKind;
  Name : ShortString;
end
```

The `TypeInfo` function returns a pointer to a `TTypeInfo` record.

Note that the Name field is stored with as much bytes as needed to store the name, it is not padded to 255 characters. The type data immediatly follows the `TTypeInfo` record as a TTypeData (1510) record.

```
TTypeKind = (tkUnknown,tkInteger,tkChar,tkEnumeration,tkFloat,tkSet,
             tkMethod,tkSString,tkLString,tkAString,tkWString,tkVariant,
             tkArray,tkRecord,tkInterface,tkClass,tkObject,tkWChar,
             tkBool,tkInt64,tkQWord,tkDynArray,tkInterfaceRaw)
```

Table 31.10: Enumeration values for type TTypeKind

Value	Explanation
tkArray	Array property.
tkAString	Ansistring property.
tkBool	Boolean property.
tkChar	Char property.
tkClass	Class property.
tkDynArray	Dynamical array property.
tkEnumeration	Enumeration type property.
tkFloat	Float property.
tkInt64	Int64 property.
tkInteger	Integer property.
tkInterface	Interface property.
tkInterfaceRaw	Raw interface property.
tkLString	Longstring property.
tkMethod	Method property.
tkObject	Object property.
tkQWord	QWord property.
tkRecord	Record property.
tkSet	Set property.
tkSString	Shortstring property.
tkUnknown	Unknown property type.
tkVariant	Variant property.
tkWChar	Widechar property.
tkWString	Widestring property.

Type of a property.

```
TTypeKinds= Set of (tkArray,tkAString,tkBool,tkChar,tkClass,tkDynArray,
                    tkEnumeration,tkFloat,tkInt64,tkInteger,tkInterface,
                    tkInterfaceRaw,tkLString,tkMethod,tkObject,tkQWord,
                    tkRecord,tkSet,tkSString,tkUnknown,tkVariant,
                    tkWChar,tkWString)
```

Set of TTypeKind (1511) enumeration.

31.7 Procedures and functions

31.7.1 FindPropInfo

Synopsis: Return property information by property name.

Declaration:
```
function FindPropInfo(Instance: TObject;const PropName: String)
                          : PPropInfo
function FindPropInfo(AClass: TClass;const PropName: String) : PPropInfo
```

Visibility: default

Description: `FindPropInfo` examines the published property information of a class and returns a pointer to the property information for property `PropName`. The class to be examined can be specified in one of two ways:

AClassa class pointer.

Instancean instance of the class to be investigated.

If the property does not exist, a `EPropertyError` exception will be raised. The GetPropInfo (1521) function has the same function as the `FindPropInfo` function, but returns `Nil` if the property does not exist.

Errors: Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetPropInfo (1521), GetPropList (1523), GetPropInfos (1522)

Listing: ./typinfex/ex14.pp

```
Program example13;

{ This program demonstrates the FindPropInfo function }

{$mode objfpc}

uses
  rttiobj,typinfo,sysutils;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PPropInfo;
  I,J : Longint;
  PP : PPropList;
  prI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=FindPropInfo(O,'BooleanField');
  WriteIn('FindPropInfo(Instance,BooleanField) : ',PI^.Name);
  PI:=FindPropInfo(O.ClassType,'ByteField');
  WriteIn('FindPropInfo(Class,ByteField)       : ',PI^.Name);
  Write  ('FindPropInfo(Class,NonExistingProp) : ');
  Try
    PI:=FindPropInfo(O,'NonExistingProp');
  except
    On E: Exception do
      WriteIn('Caught exception "',E.ClassName,'" with message : ',E.Message);
  end;
  O.Free;
end.
```

31.7.2 GetEnumName

Synopsis: Return name of enumeration constant.

Declaration: `function GetEnumName(TypeInfo: PTypeInfo;Value: Integer) : String`

Visibility: default

Description: `GetEnumName` scans the type information for the enumeration type described by `TypeInfo` and returns the name of the enumeration constant for the element with ordinal value equal to `Value`.

If `Value` is out of range, the first element of the enumeration type is returned. The result is lowercased, but this may change in the future.

This can be used in combination with `GetOrdProp` to stream a property of an enumerated type.

Errors: No check is done to determine whether `TypeInfo` really points to the type information for an enumerated type.

See also: GetOrdProp (1520), GetEnumValue (1514)

Listing: ./typinfex/ex9.pp

```
program example9;

{ This program demonstrates the GetEnumName, GetEnumValue functions }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  TI : PTypeInfo;

begin
  O:=TMyTestObject.Create;
  TI:=GetPropInfo(O,'MyEnumField')^.PropType;
  Writeln('GetEnumName              : ',GetEnumName(TI,Ord(O.MyEnumField)));
  Writeln('GetEnumValue(mefirst) : ',GetEnumName(TI,GetEnumValue(TI,'mefirst')));
  O.Free;
end.
```

31.7.3 GetEnumNameCount

Synopsis: Return number of names in an enumerated type

Declaration: `function GetEnumNameCount(enum1: PTypeInfo) : SizeInt`

Visibility: default

Description: `GetEnumNameCount` returns the number of values (names) in the enumerated type, described by `enum1`

Errors: No checking is done to see whether `Enum1` is really type information of an enumerated type.

See also: GetEnumValue (1514), GetEnumName (1512)

31.7.4 GetEnumProp

Synopsis: Return the value of an enumeration type property.

Declaration:
```
function GetEnumProp(Instance: TObject;const PropName: String) : String
function GetEnumProp(Instance: TObject;const PropInfo: PPropInfo)
                     : String
```

Visibility: default

Description: `GetEnumProp` returns the value of an property of an enumerated type and returns the name of the enumerated value for the objetc `Instance`. The property whose value must be returned can be specified by its property info in `PropInfo` or by its name in `PropName`

Errors: No check is done to determine whether `PropInfo` really points to the property information for an enumerated type. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetEnumProp (1531), GetOrdProp (1520), GetStrProp (1525), GetInt64Prop (1516), GetMethodProp (1517), GetSetProp (1524), GetObjectProp (1519), GetEnumProp (1513)

Listing: ./typinfex/ex2.pp

```
program example2;

{ This program demonstrates the GetEnumProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  TI : PTypeInfo;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'MyEnumField');
  TI:=PI^.PropType;
  Writeln('Enum property     : ');
  Writeln('Value                    : ',GetEnumName(TI,Ord(O.MyEnumField)));
  Writeln('Get (name)               : ',GetEnumProp(O,'MyEnumField'));
  Writeln('Get (propinfo)           : ',GetEnumProp(O,PI));
  SetEnumProp(O,'MyEnumField','meFirst');
  Writeln('Set (name,meFirst)       : ',GetEnumName(TI,Ord(O.MyEnumField)));
  SetEnumProp(O,PI,'meSecond');
  Writeln('Set (propinfo,meSecond) : ',GetEnumName(TI,Ord(O.MyEnumField)));
  O.Free;
end.
```

31.7.5 GetEnumValue

Synopsis: Get ordinal value for enumerated type by name

Declaration: `function GetEnumValue(TypeInfo: PTypeInfo;const Name: String) : Integer`

Visibility: default

Description: `GetEnumValue` scans the type information for the enumeration type described by `TypeInfo` and returns the ordinal value for the element in the enumerated type that has identifier `Name`. The identifier is searched in a case-insensitive manner.

This can be used to set the value of enumerated properties from a stream.

For an example, see GetEnumName (1512).

Errors: If `Name` is not found in the list of enumerated values, then -1 is returned. No check is done whether `TypeInfo` points to the type information for an enumerated type.

See also: GetEnumName (1512), SetOrdProp (1533)

31.7.6 GetFloatProp

Synopsis: Return value of floating point property

Declaration:
```
function GetFloatProp(Instance: TObject;PropInfo: PPropInfo) : Extended
function GetFloatProp(Instance: TObject;const PropName: String)
                      : Extended
```

Visibility: default

Description: `GetFloatProp` returns the value of the float property described by `PropInfo` or with name `Propname` for the object `Instance`. All float types are converted to extended.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetFloatProp (1531), GetOrdProp (1520), GetStrProp (1525), GetInt64Prop (1516), GetMethodProp (1517), GetSetProp (1524), GetObjectProp (1519), GetEnumProp (1513)

Listing: ./typinfex/ex4.pp

```
program example4;

{ This program demonstrates the GetFloatProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Real property : ');
  PI:=GetPropInfo(O,'RealField');
  Writeln('Value              : ',O.RealField);
  Writeln('Get (name)         : ',GetFloatProp(O,'RealField'));
  Writeln('Get (propinfo)     : ',GetFloatProp(O,PI));
  SetFloatProp(O,'RealField',system.Pi);
  Writeln('Set (name,pi)      : ',O.RealField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e)   : ',O.RealField);
  Writeln('Extended property : ');
  PI:=GetPropInfo(O,'ExtendedField');
  Writeln('Value              : ',O.ExtendedField);
  Writeln('Get (name)         : ',GetFloatProp(O,'ExtendedField'));
  Writeln('Get (propinfo)     : ',GetFloatProp(O,PI));
  SetFloatProp(O,'ExtendedField',system.Pi);
  Writeln('Set (name,pi)      : ',O.ExtendedField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e)   : ',O.ExtendedField);
  O.Free;
end.
```

31.7.7 GetInt64Prop

Synopsis: return value of an Int64 property

Declaration:
```
function GetInt64Prop(Instance: TObject;PropInfo: PPropInfo) : Int64
function GetInt64Prop(Instance: TObject;const PropName: String) : Int64
```

Visibility: default

Description: Publishing of Int64 properties is not yet supported by Free Pascal. This function is provided for Delphi compatibility only at the moment.

`GetInt64Prop` returns the value of the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception

See also: SetInt64Prop (1531), GetOrdProp (1520), GetStrProp (1525), GetFloatProp (1515), GetMethodProp (1517), GetSetProp (1524), GetObjectProp (1519), GetEnumProp (1513)

Listing: ./typinfex/ex15.pp

```
program example15;

{ This program demonstrates the GetInt64Prop function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Int64 property : ');
  PI:=GetPropInfo(O,'Int64Field');
  Writeln('Value              : ',O.Int64Field);
  Writeln('Get (name)         : ',GetInt64Prop(O,'Int64Field'));
  Writeln('Get (propinfo)     : ',GetInt64Prop(O,PI));
  SetInt64Prop(O,'Int64Field',12345);
  Writeln('Set (name,12345)     : ',O.Int64Field);
  SetInt64Prop(O,PI,54321);
  Writeln('Set (propinfo,54321) : ',O.Int64Field);
  O.Free;
end.
```

31.7.8 GetInterfaceProp

Synopsis: Return interface-typed property

Declaration:
```
function GetInterfaceProp(Instance: TObject;const PropName: String)
                          : IInterface
function GetInterfaceProp(Instance: TObject;PropInfo: PPropInfo)
                          : IInterface
```

Visibility: default

Description: `GetInterfaceProp` returns the interface which the property described by `PropInfo` or with name `Propname` points to for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetInterfaceProp (1532), GetOrdProp (1520), GetStrProp (1525), GetFloatProp (1515), GetInt64Prop (1516), GetSetProp (1524), GetObjectProp (1519), GetEnumProp (1513)

31.7.9 GetMethodProp

Synopsis: Return value of a method property

Declaration:
```
function GetMethodProp(Instance: TObject;PropInfo: PPropInfo) : TMethod
function GetMethodProp(Instance: TObject;const PropName: String)
                       : TMethod
```

Visibility: default

Description: `GetMethodProp` returns the method the property described by `PropInfo` or with name `Propname` for object `Instance`. The return type `TMethod` is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

`Data` points to the instance of the class with the method `Code`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (1532), GetOrdProp (1520), GetStrProp (1525), GetFloatProp (1515), GetInt64Prop (1516), GetSetProp (1524), GetObjectProp (1519), GetEnumProp (1513)

Listing: ./typinfex/ex6.pp

```
program example6;

{ This program demonstrates the GetMethodProp function }

{$mode objfpc}

uses rttiobj,typinfo,sysutils;

Type
  TNotifyObject = Class(TObject)
    Procedure Notification1(Sender : TObject);
    Procedure Notification2(Sender : TObject);
  end;

Procedure TNotifyObject.Notification1(Sender : TObject);

begin
  Write('Received notification 1 of object with class: ');
```

```
  WriteLn(Sender.ClassName);
end;

Procedure TNotifyObject.Notification2(Sender : TObject);

begin
  Write('Received notification 2 of object with class: ');
  WriteLn(Sender.ClassName);
end;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO : TNotifyObject;
  M : TMethod;

Procedure PrintMethod (Const M : TMethod);

begin
  If (M.Data=Pointer(NO)) Then
    If (M.Code=Pointer(@TNotifyObject.Notification1)) then
      WriteLn('Notification1')
    else If (M.Code=Pointer(@TNotifyObject.Notification2)) then
      WriteLn('Notification2')
    else
      begin
      Write('Unknown method adress (data:');
      Write(hexStr(Longint(M.data),8));
      WriteLn(',code:',hexstr(Longint(M.Code),8),')');
      end;
end;


begin
  O:=TMyTestObject.Create;
  NO:=TNotifyObject.Create;
  O.NotifyEvent:=@NO.Notification1;
  PI:=GetPropInfo(O,'NotifyEvent');
  WriteLn('Method property : ');
  Write('Notifying                    : ');
  O.Notify;
  Write('Get (name)                   : ');
  M:=GetMethodProp(O,'NotifyEvent');
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  Write('Get (propinfo)               : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  M:=TMethod(@NO.Notification2);
  SetMethodProp(O,'NotifyEvent',M);
  Write('Set (name,Notification2)     : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  Write('Set (propinfo,Notification1) : ');
  M:=TMethod(@NO.Notification1);
```

```
  SetMethodProp(O,PI,M);
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  O.Free;
end.
```

31.7.10 GetObjectProp

Synopsis: Return value of an object-type property.

Declaration:
```
function GetObjectProp(Instance: TObject;const PropName: String)
                        : TObject
function GetObjectProp(Instance: TObject;const PropName: String;
                       MinClass: TClass) : TObject
function GetObjectProp(Instance: TObject;PropInfo: PPropInfo) : TObject
function GetObjectProp(Instance: TObject;PropInfo: PPropInfo;
                       MinClass: TClass) : TObject
```

Visibility: default

Description: `GetObjectProp` returns the object which the property described by `PropInfo` with name `Propname` points to for object `Instance`.

If `MinClass` is specified, then if the object is not descendent of class `MinClass`, then `Nil` is returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (1532), GetOrdProp (1520), GetStrProp (1525), GetFloatProp (1515), GetInt64Prop (1516), GetSetProp (1524), GetObjectProp (1519), GetEnumProp (1513)

Listing: ./typinfex/ex5.pp

```
program example5;

{ This program demonstrates the GetObjectProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO1,NO2 : TNamedObject;

begin
  O:=TMyTestObject.Create;
  NO1:=TNamedObject.Create;
  NO1.ObjectName:='First named object';
  NO2:=TNamedObject.Create;
  NO2.ObjectName:='Second named object';
  O.ObjField:=NO1;
```

```
  WriteLn('Object property : ');
  PI:=GetPropInfo(O,'ObjField');
  Write('Property class      : ');
  WriteLn(GetObjectPropClass(O,'ObjField').ClassName);
  Write('Value               : ');
  WriteLn((O.ObjField as TNamedObject).ObjectName);
  Write('Get (name)          : ');
  WriteLn((GetObjectProp(O,'ObjField') As TNamedObject).ObjectName);
  Write('Get (propinfo)      : ');
  WriteLn((GetObjectProp(O,PI,TObject) as TNamedObject).ObjectName);
  SetObjectProp(O,'ObjField',NO2);
  Write('Set (name,NO2)      : ');
  WriteLn((O.ObjField as TNamedObject).ObjectName);
  SetObjectProp(O,PI,NO1);
  Write('Set (propinfo,NO1) : ');
  WriteLn((O.ObjField as TNamedObject).ObjectName);
  O.Free;
end.
```

31.7.11 GetObjectPropClass

Synopsis: Return class of property.

Declaration:
```
function GetObjectPropClass(Instance: TObject;const PropName: String)
                                  : TClass
```

Visibility: default

Description: `GetObjectPropClass` returns the declared class of the property with name `PropName`. This may not be the actual class of the property value.

For an example, see GetObjectProp (1519).

Errors: No checking is done whether `Instance` is non-nil. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (1532), GetOrdProp (1520), GetStrProp (1525), GetFloatProp (1515), GetInt64Prop (1516)

31.7.12 GetOrdProp

Synopsis: Get the value of an ordinal property

Declaration:
```
function GetOrdProp(Instance: TObject;PropInfo: PPropInfo) : Int64
function GetOrdProp(Instance: TObject;const PropName: String) : Int64
```

Visibility: default

Description: `GetOrdProp` returns the value of the ordinal property described by `PropInfo` or with name `PropName` for the object `Instance`. The value is returned as a longint, which should be typecasted to the needed type.

Ordinal properties that can be retrieved include:

Integers and subranges of integers The value of the integer will be returned.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type will be returned.

SetsIf the base type of the set has less than 31 possible values. If a bit is set in the return value, then the corresponding element of the base ordinal class of the set type must be included in the set.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance` Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetOrdProp (1533), GetStrProp (1525), GetFloatProp (1515), GetInt64Prop (1516), GetMethodProp (1517), GetSetProp (1524), GetObjectProp (1519), GetEnumProp (1513)

Listing: ./typinfex/ex1.pp

```
program example1;

{ This program demonstrates the GetOrdProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Boolean property     : ');
  Writeln('Value                : ',O.BooleanField);
  Writeln('Ord(Value)           : ',Ord(O.BooleanField));
  Writeln('Get (name)           : ',GetOrdProp(O,'BooleanField'));
  PI:=GetPropInfo(O,'BooleanField');
  Writeln('Get (propinfo)       : ',GetOrdProp(O,PI));
  SetOrdProp(O,'BooleanField',Ord(False));
  Writeln('Set (name,false)     : ',O.BooleanField);
  SetOrdProp(O,PI,Ord(True));
  Writeln('Set (propinfo,true) : ',O.BooleanField);
  O.Free;
end.
```

31.7.13 GetPropInfo

Synopsis: Return property type information, by property name.

Declaration:
```
function GetPropInfo(TypeInfo: PTypeInfo;const PropName: String)
                     : PPropInfo
function GetPropInfo(TypeInfo: PTypeInfo;const PropName: String;
                    AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(Instance: TObject;const PropName: String;
                    AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(Instance: TObject;const PropName: String)
                     : PPropInfo
function GetPropInfo(AClass: TClass;const PropName: String;
                    AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(AClass: TClass;const PropName: String) : PPropInfo
```

Visibility: default

Description: `GetPropInfo` returns a pointer to the `TPropInfo` record for a the `PropName` property of a class. The class to examine can be specified in one of three ways:

InstanceAn instance of the class.

AClassA class pointer to the class.

TypeInfoA pointer to the type information of the class.

In each of these three ways, if `AKinds` is specified, if the property has `TypeKind` which is not included in `Akinds`, `Nil` will be returned.

For an example, see most of the other functions.

Errors: If the property `PropName` does not exist, `Nil` is returned.

See also: GetPropInfos (1522), GetPropList (1523)

31.7.14 GetPropInfos

Synopsis: Return a list of published properties.

Declaration: `procedure GetPropInfos(TypeInfo: PTypeInfo;PropList: PPropList)`

Visibility: default

Description: `GetPropInfos` stores pointers to the property information of all published properties of a class with class info `TypeInfo` in the list pointed to by `Proplist`. The `PropList` pointer must point to a memory location that contains enough space to hold all properties of the class and its parent classes.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: GetPropInfo (1521), GetPropList (1523)

Listing: ./typinfex/ex12.pp

```
Program example12;

{ This program demonstrates the GetPropInfos function }

uses
  rttiobj,typinfo;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PTypeInfo;
  I,J : Longint;
  PP : PPropList;
  prl : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=O.ClassInfo;
  PT:=GetTypeData(PI);
  Writeln('Property Count : ',PT^.PropCount);
  GetMem (PP,PT^.PropCount*SizeOf(Pointer));
```

```
  GetPropInfos(PI,PP);
  For I:=0 to PT^.PropCount-1 do
    begin
    With PP^[i]^ do
      begin
      Write('Property ',i+1:3,': ',name:30);
      writeln('  Type: ',TypeNames[typinfo.PropType(O,Name)]);
      end;
    end;
  FreeMem(PP);
  O.Free;
end.
```

31.7.15 GetPropList

Synopsis: Return a list of a certain type of published properties.

Declaration:
```
function GetPropList(TypeInfo: PTypeInfo;TypeKinds: TTypeKinds;
                     PropList: PPropList;Sorted: Boolean) : LongInt
function GetPropList(TypeInfo: PTypeInfo;out PropList: PPropList)
                     : SizeInt
function GetPropList(AObject: TObject;out PropList: PPropList) : Integer
```

Visibility: default

Description: `GetPropList` stores pointers to property information of the class with class info `TypeInfo` for properties of kind `TypeKinds` in the list pointed to by `Proplist`. `PropList` must contain enough space to hold all properties.

The function returns the number of pointers that matched the criteria and were stored in `PropList`.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: GetPropInfos (1522), GetPropInfo (1521)

Listing: ./typinfex/ex13.pp

```
Program example13;

{ This program demonstrates the GetPropList function }

uses
  rttiobj,typinfo;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PTypeInfo;
  I,J : Longint;
  PP : PPropList;
  prl : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=O.ClassInfo;
  PT:=GetTypeData(PI);
```

```
  Writeln('Total property Count : ',PT^.PropCount);
  GetMem (PP,PT^.PropCount*SizeOf(Pointer));
  J:=GetPropList(PI,OrdinalTypes,PP);
  Writeln('Ordinal property Count : ',J);
  For I:=0 to J-1 do
    begin
    With PP^[i]^ do
      begin
      Write('Property ',i+1:3,': ',name:30);
      writeln('  Type: ',TypeNames[typinfo.PropType(O,Name)]);
      end;
    end;
  FreeMem(PP);
  O.Free;
end.
```

31.7.16 GetPropValue

Synopsis: Get property value as a string.

Declaration:
```
function GetPropValue(Instance: TObject;const PropName: String)
                        : Variant
function GetPropValue(Instance: TObject;const PropName: String;
                      PreferStrings: Boolean) : Variant
```

Visibility: default

Description: Due to missing `Variant` support, `GetPropValue` is not yet implemented. The declaration is provided for compatibility with Delphi.

Errors:

31.7.17 GetSetProp

Synopsis: Return the value of a set property.

Declaration:
```
function GetSetProp(Instance: TObject;const PropName: String) : String
function GetSetProp(Instance: TObject;const PropName: String;
                    Brackets: Boolean) : String
function GetSetProp(Instance: TObject;const PropInfo: PPropInfo;
                    Brackets: Boolean) : String
```

Visibility: default

Description: `GetSetProp` returns the contents of a set property as a string. The property to be returned can be specified by it's name in `PropName` or by its property information in `PropInfo`.

The returned set is a string representation of the elements in the set as returned by SetToString (1535). The `Brackets` option can be used to enclose the string representation in square brackets.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance` Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetSetProp (1534), GetStrProp (1525), GetFloatProp (1515), GetInt64Prop (1516), GetMethodProp (1517)

Listing: ./typinfex/ex7.pp

```
program example7;

{ This program demonstrates the GetSetProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

Function SetAsString (ASet : TMyEnums) : String;

Var
  i : TmyEnum;

begin
  result:='';
  For i:=mefirst to methird do
    If i in ASet then
      begin
      If (Result<>'') then
        Result:=Result+',';
      Result:=Result+MyEnumNames[i];
      end;
end;

Var
  S : TMyEnums;

begin
  O:=TMyTestObject.Create;
  O.SetField:=[mefirst,meSecond,meThird];
  Writeln('Set property     : ');
  Writeln('Value                    : ',SetAsString(O.SetField));
  Writeln('Ord(Value)               : ',Longint(O.SetField));
  Writeln('Get (name)               : ',GetSetProp(O,'SetField'));
  PI:=GetPropInfo(O,'SetField');
  Writeln('Get (propinfo)           : ',GetSetProp(O,PI,false));
  S:=[meFirst,meThird];
  SetOrdProp(O,'SetField',Integer(S));
  Write('Set (name,[mefirst,methird]) : ');
  Writeln(SetAsString(O.SetField));
  S:=[meSecond];
  SetOrdProp(O,PI,Integer(S));
  Write('Set (propinfo,[meSecond])    : ');
  Writeln(SetAsString(O.SetField));
  O.Free;
end.
```

31.7.18 GetStrProp

Synopsis: Return the value of a string property.

Declaration: `function GetStrProp(Instance: TObject;PropInfo: PPropInfo) : Ansistring`

```
function GetStrProp(Instance: TObject;const PropName: String) : String
```

Visibility: default

Description: `GetStrProp` returns the value of the string property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetStrProp (1534), SetWideStrProp (1536), GetOrdProp (1520), GetFloatProp (1515), GetInt64Prop (1516), GetMethodProp (1517)

Listing: ./typinfex/ex3.pp

```
program example3;

{ This program demonstrates the GetStrProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'AnsiStringField');
  Writeln('String property :');
  Writeln('Value                    : ',O.AnsiStringField);
  Writeln('Get (name)               : ',GetStrProp(O,'AnsiStringField'));
  Writeln('Get (propinfo)           : ',GetStrProp(O,PI));
  SetStrProp(O,'AnsiStringField','First');
  Writeln('Set (name,''First'')       : ',O.AnsiStringField);
  SetStrProp(O,PI,'Second');
  Writeln('Set (propinfo,''Second'') : ',O.AnsiStringField);
  O.Free;
end.
```

31.7.19 GetTypeData

Synopsis: Return a pointer to type data, based on type information.

Declaration: `function GetTypeData(TypeInfo: PTypeInfo) : PTypeData`

Visibility: default

Description: `GetTypeData` returns a pointer to the `TTypeData` record that follows after the `TTypeInfo` record pointed to by `TypeInfo`. It essentially skips the `Kind` and `Name` fields in the `TTypeInfo` record.

Errors: None.

31.7.20 GetVariantProp

Synopsis: Return the value of a variant property.

Declaration:
```
function GetVariantProp(Instance: TObject;PropInfo: PPropInfo) : Variant
function GetVariantProp(Instance: TObject;const PropName: String)
                        : Variant
```

Visibility: default

Description: Due to mising Variant support, the `GetVariantProp` function is not yet implemented. Provided for Delphi compatibility only.

Errors:

See also: SetVariantProp (1536)

31.7.21 GetWideStrProp

Synopsis: Read a widestring property

Declaration:
```
function GetWideStrProp(Instance: TObject;PropInfo: PPropInfo)
                        : WideString
function GetWideStrProp(Instance: TObject;const PropName: String)
                        : WideString
```

Visibility: default

Description: `GetWideStrProp` returns the value of the widestring property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetStrProp (1525), SetWideStrProp (1536), GetOrdProp (1520), GetFloatProp (1515), GetInt64Prop (1516), GetMethodProp (1517)

31.7.22 IsPublishedProp

Synopsis: Check whether a published property exists.

Declaration:
```
function IsPublishedProp(Instance: TObject;const PropName: String)
                         : Boolean
function IsPublishedProp(AClass: TClass;const PropName: String)
                         : Boolean
```

Visibility: default

Description: `IsPublishedProp` returns true if a class has a published property with name `PropName`. The class can be specfied in one of two ways:

AClassA class pointer to the class.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsStoredProp (1528), PropIsType (1529)

Listing: ./typinfex/ex10.pp

```
program example10;

{ This program demonstrates the IsPublishedProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('IsPublishedProp(O,BooleanField)      : ');
  Writeln(IsPublishedProp(O,'BooleanField'));
  Write('IsPublishedProp(Class,BooleanField) : ');
  Writeln(IsPublishedProp(O.ClassType,'BooleanField'));
  Write('IsPublishedProp(O,SomeField)         : ');
  Writeln(IsPublishedProp(O,'SomeField'));
  Write('IsPublishedProp(Class,SomeField)     : ');
  Writeln(IsPublishedProp(O.ClassType,'SomeField'));
  O.Free;
end.
```

31.7.23 IsStoredProp

Synopsis: Check whether a property is stored.

Declaration:
```
function IsStoredProp(Instance: TObject;PropInfo: PPropInfo) : Boolean
function IsStoredProp(Instance: TObject;const PropName: String)
                      : Boolean
```

Visibility: default

Description: `IsStoredProp` returns `True` if the `Stored` modifier evaluates to `True` for the property described by `PropInfo` or with name `PropName` for object `Instance`. It returns `False` otherwise. If the function returns `True`, this indicates that the property should be written when streaming the object `Instance`.

If there was no `stored` modifier in the declaration of the property, `True` will be returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (1527), PropIsType (1529)

Listing: ./typinfex/ex11.pp

```
program example11;

{ This program demonstrates the IsStoredProp function }
```

```
{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Stored tests     : ');
  Write('IsStoredProp(O,StoredIntegerConstFalse)     : ');
  Writeln(IsStoredProp(O,'StoredIntegerConstFalse'));
  Write('IsStoredProp(O,StoredIntegerConstTrue)      : ');
  Writeln(IsStoredProp(O,'StoredIntegerConstTrue'));
  Write('IsStoredProp(O,StoredIntegerMethod)         : ');
  Writeln(IsStoredProp(O,'StoredIntegerMethod'));
  Write('IsStoredProp(O,StoredIntegerVirtualMethod) : ');
  Writeln(IsStoredProp(O,'StoredIntegerVirtualMethod'));
  O.Free;
end.
```

31.7.24 PropIsType

Synopsis: Check the type of a published property.

Declaration:
```
function PropIsType(Instance: TObject;const PropName: String;
                    TypeKind: TTypeKind) : Boolean
function PropIsType(AClass: TClass;const PropName: String;
                    TypeKind: TTypeKind) : Boolean
```

Visibility: default

Description: `PropIsType` returns `True` if the property with name `PropName` has type `TypeKind`. It returns `False` otherwise. The class to be examined can be specified in one of two ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers.Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (1527), IsStoredProp (1528), PropType (1530)

Listing: ./typinfex/ex16.pp

```
program example16;

{ This program demonstrates the PropIsType function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
```

```
begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('PropIsType(O,BooleanField,tkBool)      : ');
  Writeln(PropIsType(O,'BooleanField',tkBool));
  Write('PropIsType(Class,BooleanField,tkBool) : ');
  Writeln(PropIsType(O.ClassType,'BooleanField',tkBool));
  Write('PropIsType(O,ByteField,tkString)       : ');
  Writeln(PropisType(O,'ByteField',tkString));
  Write('PropIsType(Class,ByteField,tkString)   : ');
  Writeln(PropIsType(O.ClassType,'ByteField',tkString));
  O.Free;
end.
```

31.7.25 PropType

Synopsis: Return the type of a property

Declaration:
```
function PropType(Instance: TObject;const PropName: String) : TTypeKind
function PropType(AClass: TClass;const PropName: String) : TTypeKind
```

Visibility: default

Description: `Proptype` returns the type of the property `PropName` for a class. The class to be examined can be specified in one of 2 ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (1527), IsStoredProp (1528), PropIsType (1529)

Listing: ./typinfex/ex17.pp

```
program example17;

{ This program demonstrates the PropType function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('PropType(O,BooleanField)      : ');
  Writeln(TypeNames[PropType(O,'BooleanField')]);
  Write('PropType(Class,BooleanField) : ');
  Writeln(TypeNames[PropType(O.ClassType,'BooleanField')]);
  Write('PropType(O,ByteField)         : ');
  Writeln(TypeNames[PropType(O,'ByteField')]);
```

```
  Write('PropType(Class,ByteField)      : ');
  WriteIn(TypeNames[PropType(O.ClassType,'ByteField')]);
  O.Free;
end.
```

31.7.26 SetEnumProp

Synopsis: Set value of an enumerated-type property

Declaration:
```
procedure SetEnumProp(Instance: TObject;const PropName: String;
                      const Value: String)
procedure SetEnumProp(Instance: TObject;const PropInfo: PPropInfo;
                      const Value: String)
```

Visibility: default

Description: `SetEnumProp` sets the property described by `PropInfo` or with name `PropName` to `Value`. `Value` must be a string with the name of the enumerate value, i.e. it can be used as an argument to GetEnumValue (1514).

For an example, see GetEnumProp (1513).

Errors: No checks are done to ensure `Instance` or `PropInfo` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetEnumProp (1513), SetStrProp (1534), SetFloatProp (1531), SetInt64Prop (1531), SetMethodProp (1532)

31.7.27 SetFloatProp

Synopsis: Set value of a float property.

Declaration:
```
procedure SetFloatProp(Instance: TObject;const PropName: String;
                       Value: Extended)
procedure SetFloatProp(Instance: TObject;PropInfo: PPropInfo;
                       Value: Extended)
```

Visibility: default

Description: `SetFloatProp` assigns `Value` to the property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetFloatProp (1515).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetFloatProp (1515), SetOrdProp (1533), SetStrProp (1534), SetInt64Prop (1531), SetMethodProp (1532)

31.7.28 SetInt64Prop

Synopsis: Set value of a Int64 property

Declaration:
```
procedure SetInt64Prop(Instance: TObject;PropInfo: PPropInfo;
                       const Value: Int64)
procedure SetInt64Prop(Instance: TObject;const PropName: String;
                       const Value: Int64)
```

Visibility: default

Description: `SetInt64Prop` assigns `Value` to the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetInt64Prop (1516).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetInt64Prop (1516), GetMethodProp (1517), SetOrdProp (1533), SetStrProp (1534), SetFloatProp (1531)

31.7.29 SetInterfaceProp

Synopsis: Set interface-valued property

Declaration:
```
procedure SetInterfaceProp(Instance: TObject;const PropName: String;
                           const Value: IInterface)
procedure SetInterfaceProp(Instance: TObject;PropInfo: PPropInfo;
                           const Value: IInterface)
```

Visibility: default

Description: `SetInterfaceProp` assigns `Value` to the the object property described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid interface property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetInterfaceProp (1516), SetObjectProp (1533), SetOrdProp (1533), SetStrProp (1534), SetFloatProp (1531), SetInt64Prop (1531), SetMethodProp (1532)

31.7.30 SetMethodProp

Synopsis: Set the value of a method property

Declaration:
```
procedure SetMethodProp(Instance: TObject;PropInfo: PPropInfo;
                        const Value: TMethod)
procedure SetMethodProp(Instance: TObject;const PropName: String;
                        const Value: TMethod)
```

Visibility: default

Description: `SetMethodProp` assigns `Value` to the method the property described by `PropInfo` or with name `Propname` for object `Instance`.

The type `TMethod` of the `Value` parameter is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

Data should point to the instance of the class with the method Code.

For an example, see GetMethodProp (1517).

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetMethodProp (1517), SetOrdProp (1533), SetStrProp (1534), SetFloatProp (1531), SetInt64Prop (1531)

31.7.31 SetObjectProp

Synopsis: Set the value of an object-type property.

Declaration:
```
procedure SetObjectProp(Instance: TObject;const PropName: String;
                        Value: TObject)
procedure SetObjectProp(Instance: TObject;PropInfo: PPropInfo;
                        Value: TObject)
```

Visibility: default

Description: SetObjectProp assigns Value to the the object property described by PropInfo or with name Propname for the object Instance.

For an example, see GetObjectProp (1519).

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid object property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetObjectProp (1519), SetOrdProp (1533), SetStrProp (1534), SetFloatProp (1531), SetInt64Prop (1531), SetMethodProp (1532)

31.7.32 SetOrdProp

Synopsis: Set value of an ordinal property

Declaration:
```
procedure SetOrdProp(Instance: TObject;PropInfo: PPropInfo;Value: Int64)
procedure SetOrdProp(Instance: TObject;const PropName: String;
                     Value: Int64)
```

Visibility: default

Description: SetOrdProp assigns Value to the the ordinal property described by PropInfo or with name Propname for the object Instance.

Ordinal properties that can be set include:

Integers and subranges of integers The actual value of the integer must be passed.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type must be passed.

Subrange typesof integers or enumerated types. Here the ordinal value must be passed.

SetsIf the base type of the set has less than 31 possible values. For each possible value; the corresponding bit of Value must be set.

For an example, see GetOrdProp (1520).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetOrdProp (1520), SetStrProp (1534), SetFloatProp (1531), SetInt64Prop (1531), SetMethodProp (1532)

31.7.33 SetPropValue

Synopsis: Set property value as variant

Declaration:
```
procedure SetPropValue(Instance: TObject;const PropName: String;
                       const Value: Variant)
```

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented; it is provided for Delphi compatibility only.

Errors:

31.7.34 SetSetProp

Synopsis: Set value of set-typed property.

Declaration:
```
procedure SetSetProp(Instance: TObject;const PropName: String;
                     const Value: String)
procedure SetSetProp(Instance: TObject;const PropInfo: PPropInfo;
                     const Value: String)
```

Visibility: default

Description: `SetSetProp` sets the property specified by `PropInfo` or `PropName` for object `Instance` to `Value`. `Value` is a string which contains a comma-separated list of values, each value being a string-representation of the enumerated value that should be included in the set. The value should be accepted by the StringToSet (1536) function.

The value can be formed using the SetToString (1535) function.

For an example, see GetSetProp (1524).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetSetProp (1524), SetOrdProp (1533), SetStrProp (1534), SetFloatProp (1531), SetInt64Prop (1531), SetMethodProp (1532), SetToString (1535), StringToSet (1536)

31.7.35 SetStrProp

Synopsis: Set value of a string property

Declaration:
```
procedure SetStrProp(Instance: TObject;const PropName: String;
                     const Value: AnsiString)
procedure SetStrProp(Instance: TObject;PropInfo: PPropInfo;
                     const Value: Ansistring)
```

Visibility: default

Description: `SetStrProp` assigns `Value` to the string property described by `PropInfo` or with name `Propname` for object `Instance`.

For an example, see GetStrProp (1525)

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetStrProp (1525), SetWideStrProp (1536), SetOrdProp (1533), SetFloatProp (1531), SetInt64Prop (1531), SetMethodProp (1532)

31.7.36 SetToString

Synopsis: Convert set to a string description

Declaration:
```
function SetToString(TypeInfo: PTypeInfo;Value: Integer;
                     Brackets: Boolean) : String
function SetToString(PropInfo: PPropInfo;Value: Integer;
                     Brackets: Boolean) : String
function SetToString(PropInfo: PPropInfo;Value: Integer) : String
```

Visibility: default

Description: `SetToString` takes an integer representation of a set (as received e.g. by `GetOrdProp`) and turns it into a string representing the elements in the set, based on the type information found in the `PropInfo` property information. By default, the string representation is not surrounded by square brackets. Setting the `Brackets` parameter to `True` will surround the string representation with brackets.

The function returns the string representation of the set.

Errors: No checking is done to see whether `PropInfo` points to valid property information.

See also: GetEnumName (1512), GetEnumValue (1514), StringToSet (1536)

Listing: ./typinfex/ex18.pp

```
program example18;

{ This program demonstrates the SetToString function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  I : longint;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'SetField');
  O.SetField:=[mefirst,meSecond,meThird];
  I:=GetOrdProp(O,PI);
  Writeln('Set property to string : ');
  Writeln('Value   : ',SetToString(PI,I,False));
  O.SetField:=[mefirst,meSecond];
```

```
  I:=GetOrdProp(O,PI);
  Writeln('Value    : ',SetToString(PI,I,True));
  I:=StringToSet(PI,'mefirst');
  SetOrdProp(O,PI,I);
  I:=GetOrdProp(O,PI);
  Writeln('Value    : ',SetToString(PI,I,False));
  I:=StringToSet(PI,'[mesecond,methird]');
  SetOrdProp(O,PI,I);
  I:=GetOrdProp(O,PI);
  Writeln('Value    : ',SetToString(PI,I,True));
  O.Free;
end.
```

31.7.37 SetVariantProp

Synopsis: Set value of a variant property

Declaration:
```
procedure SetVariantProp(Instance: TObject;const PropName: String;
                         const Value: Variant)
procedure SetVariantProp(Instance: TObject;PropInfo: PPropInfo;
                         const Value: Variant)
```

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented. Provided for Delphi compatibility only.

Errors:

31.7.38 SetWideStrProp

Synopsis: Set a widestring property

Declaration:
```
procedure SetWideStrProp(Instance: TObject;const PropName: String;
                         const Value: WideString)
procedure SetWideStrProp(Instance: TObject;PropInfo: PPropInfo;
                         const Value: WideString)
```

Visibility: default

Description: `SetWideStrProp` assigns `Value` to the widestring property described by `PropInfo` or with name `Propname` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetWideStrProp (1527), SetStrProp (1534), SetOrdProp (1533), SetFloatProp (1531), SetInt64Prop (1531), SetMethodProp (1532)

31.7.39 StringToSet

Synopsis: Convert string description to a set.

Declaration:
```
function StringToSet(PropInfo: PPropInfo;const Value: String) : Integer
function StringToSet(TypeInfo: PTypeInfo;const Value: String) : Integer
```

Visibility: default

Description: `StringToSet` converts the string representation of a set in `Value` to a integer representation of the set, using the property information found in `PropInfo`. This property information should point to the property information of a set property. The function returns the integer representation of the set. (i.e, the set value, typecast to an integer)

The string representation can be surrounded with square brackets, and must consist of the names of the elements of the base type of the set. The base type of the set should be an enumerated type. The elements should be separated by commas, and may be surrounded by spaces. each of the names will be fed to the GetEnumValue (1514) function.

For an example, see SetToString (1535).

Errors: No checking is done to see whether `PropInfo` points to valid property information. If a wrong name is given for an enumerated value, then an `EPropertyError` will be raised.

See also: GetEnumName (1512), GetEnumValue (1514), SetToString (1535)

31.8 EPropertyConvertError

31.8.1 Description

`EPropertyConvertError` is not used in the Free Pascal implementation of the typinfo unit, but is declared for Delphi compatibility.

31.9 EPropertyError

31.9.1 Description

Exception raised in case of an error in one of the functions.

Chapter 32

Reference for unit 'Unix'

32.1 Used units

Table 32.1: Used units by unit 'Unix'

Name	Page
BaseUnix	94
unixtype	1575

32.2 Constants, types and variables

32.2.1 Constants

`ARG_MAX = UnixType.ARG_MAX`

Maximum number of arguments to a program.

`fs_ext = $137d`

File system type (StatFS (1572)): (ext) Extended

`fs_ext2 = $ef53`

File system type (StatFS (1572)): (ext2) Second extended

`fs_iso = $9660`

File system type (StatFS (1572)): ISO 9660

`fs_minix = $137f`

File system type (StatFS (1572)): Minix

`fs_minix_30 = $138f`

File system type (StatFS (1572)): Minix 3.0

```
fs_minux_V2 = $2468
```

File system type (StatFS (1572)): Minix V2

```
fs_msdos = $4d44
```

File system type (StatFS (1572)): MSDOS (FAT)

```
fs_nfs = $6969
```

File system type (StatFS (1572)): NFS

```
fs_old_ext2 = $ef51
```

File system type (StatFS (1572)): (ext2) Old second extended

```
fs_proc = $9fa0
```

File system type (StatFS (1572)): PROC fs

```
fs_xia = $012FD16D
```

File system type (StatFS (1572)): XIA

```
IOCtl_TCGETS = $5401
```

IOCTL call number: get Terminal Control settings

```
LOCK_EX = 2
```

FpFLock (1562) Exclusive lock

```
LOCK_NB = 4
```

FpFLock (1562) Non-blocking operation

```
LOCK_SH = 1
```

FpFLock (1562) Shared lock

```
LOCK_UN = 8
```

FpFLock (1562) unlock

```
MAP_FAILED = baseunix.MAP_FAILED
```

Error return value for mmap: mmap operation failed.

```
MAP_FIXED = baseunix.MAP_FIXED
```

FpMMap (1538) map type: Interpret addr exactly

```
MAP_PRIVATE = baseunix.MAP_PRIVATE
```

FpMMap (1538) map type: Changes are private

```
MAP_SHARED = baseunix.MAP_SHARED
```

FpMMap (1538) map type: Share changes

```
MAP_TYPE = baseunix.MAP_TYPE
```

FpMMap (1538) map type: Bitmask for type of mapping

```
MS_ASYNC = 1
```

Asynchronous operation flag for msync call

```
MS_INVALIDATE = 2
```

Invalidate other mappings of file flag for msync call

```
MS_SYNC = 4
```

Synchronous operation flag for msync call

```
NAME_MAX = UnixType.NAME_MAX
```

Maximum filename length.

```
Open_Accmode = 3
```

Bitmask to determine access mode in open flags.

```
Open_Append = 2 shl 9
```

File open mode: Append to file

```
Open_Creat = 1 shl 6
```

File open mode: Create if file does not yet exist.

```
Open_Direct = 4 shl 12
```

File open mode: Minimize caching effects

```
Open_Directory = 2 shl 15
```

File open mode: File must be directory.

```
Open_Excl = 2 shl 6
```

File open mode: Open exclusively

```
Open_LargeFile = 1 shl 15
```

File open mode: Open for 64-bit I/O

```
Open_NDelay = Open_NonBlock
```

File open mode: Alias for Open_NonBlock (1541)

```
Open_NoCtty = 4 shl 6
```

File open mode: No TTY control.

```
Open_NoFollow = 4 shl 15
```

File open mode: Fail if file is symbolic link.

```
Open_NonBlock = 4 shl 9
```

File open mode: Open in non-blocking mode

```
Open_RdOnly = 0
```

File open mode: Read only

```
Open_RdWr = 2
```

File open mode: Read/Write

```
Open_Sync = 1 shl 12
```

File open mode: Write to disc at once

```
Open_Trunc = 1 shl 9
```

File open mode: Truncate file to length 0

```
Open_WrOnly = 1
```

File open mode: Write only

```
PATH_MAX = UnixType.PATH_MAX
```

Maximum pathname length.

```
PRIO_PGRP = UnixType.PRIO_PGRP
```

fpGetPriority (1538) option: Get process group priority.

```
PRIO_PROCESS = UnixType.PRIO_PROCESS
```

fpGetPriority (1538) option: Get process priority.

```
PRIO_USER = UnixType.PRIO_USER
```

fpGetPriority (1538) option: Get user priority.

```
PROT_EXEC = baseunix.PROT_EXEC
```

FpMMap (1538) memory access: page can be executed

```
PROT_NONE = baseunix.PROT_NONE
```

FpMMap (1538) memory access: page can not be accessed

```
PROT_READ = baseunix.PROT_READ
```

FpMMap (1538) memory access: page can be read

```
PROT_WRITE = baseunix.PROT_WRITE
```

FpMMap (1538) memory access: page can be written

```
P_IN = 1
```

Input file descriptor of pipe pair.

```
P_OUT = 2
```

Output file descriptor of pipe pair.

```
SIG_MAXSIG = UnixType.SIG_MAXSIG
```

Maximum system signal number.

```
STAT_IFBLK = $6000
```

File (#rtl.baseunix.stat (126) record) mode: Block device

```
STAT_IFCHR = $2000
```

File (#rtl.baseunix.stat (126) record) mode: Character device

```
STAT_IFDIR = $4000
```

File (#rtl.baseunix.stat (126) record) mode: Directory

```
STAT_IFIFO = $1000
```

File (#rtl.baseunix.stat (126) record) mode: FIFO

```
STAT_IFLNK = $a000
```

File (#rtl.baseunix.stat (126) record) mode: Link

```
STAT_IFMT = $f000
```

File (#rtl.baseunix.stat (126) record) mode: File type bit mask

```
STAT_IFREG = $8000
```

File (#rtl.baseunix.stat (126) record) mode: Regular file

```
STAT_IFSOCK = $c000
```

File (#rtl.baseunix.stat (126) record) mode: Socket

```
STAT_IRGRP = STAT_IROTH shl 3
```

File (#rtl.baseunix.stat (126) record) mode: Group read permission

```
STAT_IROTH = $4
```

File (#rtl.baseunix.stat (126) record) mode: Other read permission

```
STAT_IRUSR = STAT_IROTH shl 6
```

File (#rtl.baseunix.stat (126) record) mode: Owner read permission

```
STAT_IRWXG = STAT_IRWXO shl 3
```

File (#rtl.baseunix.stat (126) record) mode: Group permission bits mask

```
STAT_IRWXO = $7
```

File (#rtl.baseunix.stat (126) record) mode: Other permission bits mask

```
STAT_IRWXU = STAT_IRWXO shl 6
```

File (#rtl.baseunix.stat (126) record) mode: Owner permission bits mask

```
STAT_ISGID = $0400
```

File (#rtl.baseunix.stat (126) record) mode: GID bit set

```
STAT_ISUID = $0800
```

File (#rtl.baseunix.stat (126) record) mode: UID bit set

```
STAT_ISVTX = $0200
```

File (#rtl.baseunix.stat (126) record) mode: Sticky bit set

```
STAT_IWGRP = STAT_IWOTH shl 3
```

File (#rtl.baseunix.stat (126) record) mode: Group write permission

```
STAT_IWOTH = $2
```

File (#rtl.baseunix.stat (126) record) mode: Other write permission

```
STAT_IWUSR = STAT_IWOTH shl 6
```

File (#rtl.baseunix.stat (126) record) mode: Owner write permission

```
STAT_IXGRP = STAT_IXOTH shl 3
```

File (#rtl.baseunix.stat (126) record) mode: Others execute permission

```
STAT_IXOTH = $1
```

File (#rtl.baseunix.stat (126) record) mode: Others execute permission

```
STAT_IXUSR = STAT_IXOTH shl 6
```

File (#rtl.baseunix.stat (126) record) mode: Others execute permission

```
SYS_NMLN = UnixType.SYS_NMLN
```

Max system name length.

```
Wait_Any = -1
```

#rtl.baseunix.fpWaitPID (181): Wait on any process

```
Wait_Clone = $80000000
```

#rtl.baseunix.fpWaitPID (181): Wait on clone processes only.

```
Wait_MyPGRP = 0
```

#rtl.baseunix.fpWaitPID (181): Wait processes from current process group

```
Wait_NoHang = 1
```

#rtl.baseunix.fpWaitPID (181): Do not wait

```
Wait_UnTraced = 2
```

#rtl.baseunix.fpWaitPID (181): Also report stopped but untraced processes

32.2.2 Types

`cbool = UnixType.cbool`

Boolean type

`cchar = UnixType.cchar`

Alias for #rtl.UnixType.cchar (1577)

`cdouble = UnixType.cdouble`

Double precision real format.

`cfloat = UnixType.cfloat`

Floating-point real format

`cint = UnixType.cint`

C type: integer (natural size)

`cint16 = UnixType.cint16`

C type: 16 bits sized, signed integer.

`cint32 = UnixType.cint32`

C type: 32 bits sized, signed integer.

`cint64 = UnixType.cint64`

C type: 64 bits sized, signed integer.

`cint8 = UnixType.cint8`

C type: 8 bits sized, signed integer.

`clock_t = UnixType.clock_t`

Clock ticks type

`clong = UnixType.clong`

C type: long signed integer (double sized)

`clongdouble = UnixType.clongdouble`

Usually translates to an extended, but is CPU dependent.

`clonglong = UnixType.clonglong`

C type: 64-bit (double long) signed integer.

`cschar = UnixType.cschar`

Signed character type

`cshort = UnixType.cshort`

C type: short signed integer (half sized)

`csigned = UnixType.csigned`

`csigned` is an alias for cint (1545).

`csint = UnixType.csint`

Signed integer

`cslong = UnixType.cslong`

The size is CPU dependent.

`cslonglong = UnixType.cslonglong`

`cslonglong` is an alias for clonglong (1546).

`csshort = UnixType.csshort`

Short signed integer type

`cuchar = UnixType.cuchar`

Alias for #rtl.UnixType.cuchar (1578)

`cuint = UnixType.cuint`

C type: unsigned integer (natural size)

`cuint16 = UnixType.cuint16`

C type: 16 bits sized, unsigned integer.

`cuint32 = UnixType.cuint32`

C type: 32 bits sized, unsigned integer.

`cuint64 = UnixType.cuint64`

C type: 64 bits sized, unsigned integer.

`cuint8 = UnixType.cuint8`

C type: 8 bits sized, unsigned integer.

```
culong = UnixType.culong
```

C type: long unsigned integer (double sized)

```
culonglong = UnixType.culonglong
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = UnixType.cunsigned
```

Alias for #rtl.unixtype.cunsigned (1579)

```
cushort = UnixType.cushort
```

C type: short unsigned integer (half sized)

```
dev_t = UnixType.dev_t
```

Device descriptor type.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
mode_t = UnixType.mode_t
```

Inode mode type.

```
nlink_t = UnixType.nlink_t
```

Number of links type.

```
off_t = UnixType.off_t
```

Offset type.

```
pcbool = UnixType.pcbool
```

Pointer to boolean type cbool (1545)

```
pcchar = UnixType.pcchar
```

Alias for #rtl.UnixType.pcchar (1580)

```
pcdouble = UnixType.pcdouble
```

Pointer to cdouble (116) type.

```
pcfloat = UnixType.pcfloat
```

Pointer to cfloat (116) type.

```
pcint = UnixType.pcint
```

Pointer to cInt (1545) type.

```
pcint16 = UnixType.pcint16
```

Pointer to 16-bit signed integer type

```
pcint32 = UnixType.pcint32
```

Pointer to signed 32-bit integer type

```
pcint64 = UnixType.pcint64
```

Pointer to signed 64-bit integer type

```
pcint8 = UnixType.pcint8
```

Pointer to 8-bits signed integer type

```
pClock = UnixType.pClock
```

Pointer to TClock (1551) type.

```
pclong = UnixType.pclong
```

Pointer to cLong (1545) type.

```
pclongdouble = UnixType.pclongdouble
```

Pointer to the long double type clongdouble (1545)

```
pclonglong = UnixType.pclonglong
```

Pointer to longlong type.

```
pcschar = UnixType.pcschar
```

Pointer to character type cschar (1546).

```
pcshort = UnixType.pcshort
```

Pointer to cShort (1546) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to signed integer type csigned (1546).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type csint (1546)

```
pcslong = UnixType.pcslong
```

Pointer ot the signed long cslong (1546)

```
pcslonglong = UnixType.pcslonglong
```

Pointer to Signed longlong type cslonglong (1546)

```
pcsshort = UnixType.pcsshort
```

Pointer to short signed integer type csshort (1546)

```
pcuchar = UnixType.pcuchar
```

Alias for #rtl.UnixType.pcuchar (1581)

```
pcuint = UnixType.pcuint
```

Pointer to cUInt (1546) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to 16-bit unsigned integer type

```
pcuint32 = UnixType.pcuint32
```

Pointer to unsigned 32-bit integer type

```
pcuint64 = UnixType.pcuint64
```

Pointer to unsigned 64-bit integer type

```
pcuint8 = UnixType.pcuint8
```

Pointer to 8-bits unsigned integer type

```
pculong = UnixType.pculong
```

Pointer to cuLong (1547) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type

```
pcunsigned = UnixType.pcunsigned
```

Alias for #rtl.unixtype.pcunsigned (1582)

```
pcushort = UnixType.pcushort
```

Pointer to cuShort (1547) type.

```
pDev = UnixType.pDev
```

Pointer to TDev (1552) type.

```
pGid = UnixType.pGid
```

Pointer to TGid (1552) type.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (1552) type.

```
pMode = UnixType.pMode
```

Pointer to TMode (1552) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (1552) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (1553) type.

```
pPid = UnixType.pPid
```

Pointer to TPid (1553) type.

```
pSize = UnixType.pSize
```

Pointer to TSize (1553) type.

```
pSize_t = UnixType.pSize_t
```

```
pSocklen = UnixType.pSocklen
```

Pointer to TSockLen (1553) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (1553) type

```
pthread_cond_t = UnixType.pthread_cond_t
```

Thread conditional variable type.

```
pthread_mutex_t = UnixType.pthread_mutex_t
```

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

Posix thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (1553) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (1552) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (1552) type.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (1552) type.

```
pUid = UnixType.pUid
```

Pointer to TUid (1553) type.

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

```
TClock = UnixType.TClock
```

Alias for clock_t (1545) type.

```
TDev = UnixType.TDev
```

Table 32.2: Enumeration values for type TFSearchOption

Value	Explanation
CurrentDirectoryFirst	Search the current directory first, before all directories in the search path.
CurrentDirectoryLast	Search the current directory last, after all directories in the search path
NoCurrentDirectory	Do not search the current directory unless it is specified in the search path.

Alias for dev_t (1547) type.

```
TFSearchOption = (NoCurrentDirectory,CurrentDirectoryFirst,
                  CurrentDirectoryLast)
```

Describes the search strategy used by FSearch (1564)

```
TGid = UnixType.TGid
```

Alias for gid_t (1547) type.

```
timespec = UnixType.timespec
```

Short time specification type.

```
timeval = UnixType.timeval
```

Time specification type.

```
time_t = UnixType.time_t
```

Time span type

```
TIno = UnixType.TIno
```

Alias for ino_t (1547) type.

```
TIOCtlRequest = UnixType.TIOCtlRequest
```

Alias for the TIOCtlRequest (1587) type in unixtypes

```
TMode = UnixType.TMode
```

Alias for mode_t (1547) type.

```
TnLink = UnixType.TnLink
```

Alias for nlink_t (1547) type.

```
TOff = UnixType.TOff
```

Alias for off_t (1547) type.

```
TPid = UnixType.TPid
```

Alias for pid_t (1550) type.

```
Tpipe = baseunix.tfildes
```

Array describing a pipe pair of filedescriptors.

```
TSize = UnixType.TSize
```

Alias for size_t (1551) type

```
TSocklen = UnixType.TSocklen
```

Alias for socklen_t (1551) type.

```
TsSize = UnixType.TsSize
```

Alias for ssize_t (1551) type

```
tstatfs = UnixType.TStatFs
```

Record describing a file system in the baseunix.fpstatfs (1538) call.

```
TTime = UnixType.TTime
```

Alias for TTime (1553) type.

```
Ttimespec = UnixType.Ttimespec
```

Alias for TimeSpec (1552) type.

```
TTimeVal = UnixType.TTimeVal
```

Alias for timeval (1552) type.

```
TUid = UnixType.TUid
```

Alias for uid_t (1553) type.

```
uid_t = UnixType.uid_t
```

User ID type

32.2.3 Variables

```
tzdaylight : Boolean
```

Indicates whether daylight savings time is active.

```
tzname : Array[boolean] of pchar
```

Timezone name.

32.3 Procedures and functions

32.3.1 AssignPipe

Synopsis: Create a set of pipe file handlers

Declaration:
```
function AssignPipe(var pipe_in: cint;var pipe_out: cint) : cint
function AssignPipe(var pipe_in: text;var pipe_out: text) : cint
function AssignPipe(var pipe_in: File;var pipe_out: File) : cint
```

Visibility: default

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`.

This call is overloaded. The in and out pipe can take three forms: an typed or untyped file, a text file or a file descriptor.

If a text file is passed then reading and writing from/to the pipe can be done through the usual `Readln(Pipe_in,...)` and `Writeln(Pipe_out,...)` procedures.

The function returns `True` if everything went succesfully, `False` otherwise.

Errors: In case the function fails and returns `False`, extended error information is returned by the FpGetErrno (145) function:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: POpen (1568), #rtl.baseunix.FpMkFifo (153)

Listing: ./unixex/ex36.pp

```
Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix,Unix;

Var pipi,pipo : Text;
    s : String;

begin
  Writeln ('Assigning Pipes.');
  If assignpipe(pipi,pipo)<>0 then
    Writeln('Error assigning pipes !',fpgeterrno);
  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.
```

32.3.2 AssignStream

Synopsis: Assign stream for in and output to a program

Declaration:
```
function AssignStream(var StreamIn: text;var Streamout: text;
                      const Prog: ansiString;
                      const args: Array of ansistring) : cint
function AssignStream(var StreamIn: text;var Streamout: text;
                      var streamerr: text;const Prog: ansiString;
                      const args: Array of ansistring) : cint
```

Visibility: default

Description: `AssignStream` creates a 2 or 3 pipes, i.e. two (or three) file objects, one for input, one for output,(and one for standard error) the other ends of these pipes are connected to standard input and output (and standard error) of `Prog`. `Prog` is the path of a program (including path). The options for the program can be specified in `Args`.

What is written to `StreamOut`, will go to the standard input of `Prog`. Whatever is written by `Prog` to it's standard output can be read from `StreamIn`. Whatever is written by `Prog` to it's standard error read from `StreamErr`, if present.

Reading and writing happens through the usual `Readln(StreamIn,...)` and `Writeln (StreamOut,...)` procedures.

Remark: You should *not* use `Reset` or `Rewrite` on a file opened with `POpen`. This will close the file before re-opening it again, thereby closing the connection with the program.

The function returns the process ID of the spawned process, or -1 in case of error.

Errors: Extended error information is returned by the FpGetErrno (145) function.

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 (1554), POpen (1568)

Listing: ./unixex/ex38.pp

```
Program Example38;

{ Program to demonstrate the AssignStream function. }

Uses BaseUnix,Unix;

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 fpgeterrno<>0 then
      begin
      writeln ('AssignStream failed !');
      halt(1);
      end;
```

```
    WriteIn ('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;
    WriteIn ('Stopped conversation');
    Close (Si);
    WriteIn ('Put down phone');
    end
  Else
    begin
    WriteIn ('This is the son ');
    While not eof (input) do
      begin
      readln (s);
      if pos ('Hello son !',S)<>0 then
         WriteIn ('Hello Dad !')
      else
         writeln ('Who are you ?');
      end;
    close (output);
    end
end.
```

32.3.3 FpExecL

Synopsis: Execute process (using argument list, environment)

Declaration:
```
function FpExecL(const PathName: AnsiString;
                 const S: Array of AnsiString) : cint
```

Visibility: default

Description: `FpExecL` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` must be an absolute pathname. The current process' environment is passed to the program. On success, `FpExecL` does not return.

Errors: Extended error information is returned by the FpGetErrno (145) function:

sys_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, 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: #rtl.baseunix.fpexecve (139), FpExecv (1559), FpExecvp (1560), FpExecle (1557), FpExeclp (1558), #rtl.baseunix.FpFork (142)

Listing: ./unixex/ex77.pp

```
Program Example77;

{ Program to demonstrate the FPExecL function. }

Uses Unix, strings;

begin
  { Execute 'ls -l', with current environment. }
  { 'ls' is NOT looked for in PATH environment variable.}
  FpExecL ('/bin/ls',['-l']);
end.
```

32.3.4 FpExecLE

Synopsis: Execute process (using argument list, environment)

Declaration:
```
function FpExecLE(const PathName: AnsiString;
                  const S: Array of AnsiString;MyEnv: ppchar) : cint
```

Visibility: default

Description: `FpExecLE` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` must be an absolute pathname. The environment in `MyEnv` is passed to the program. On success, `FpExecLE` does not return.

Errors: Extended error information is returned by the FpGetErrno (145) function:

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: #rtl.baseunix.fpexecve (139), FpExecv (1559), FpExecvp (1560), FpExecl (1556), FpExeclp (1558), #rtl.baseunix.FpFork (142)

Listing: ./unixex/ex11.pp

```
Program Example11;

{ Program to demonstrate the Execle function. }

Uses Unix, 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.
```

32.3.5 FpExecLP

Synopsis: Execute process (using argument list, environment; search path)

Declaration:
```
function FpExecLP(const PathName: AnsiString;
                  const S: Array of AnsiString) : cint
```

Visibility: default

Description: `FpExecLP` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` is searched in the path, if it isn't an absolute filename. The current environment is passed to the program. On success, `FpExecLP` does not return.

Errors: Extended error information is returned by the FpGetErrno (145) function:

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: #rtl.baseunix.fpexecve (139), FpExecv (1559), FpExecvp (1560), FpExecle (1557), FpExecl (1556), #rtl.baseunix.FpFork (142)

Listing: ./unixex/ex76.pp

```
Program Example76;

{ Program to demonstrate the FpExeclp function. }

Uses Unix, strings;

begin
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable.}
  { envp is defined in the system unit.}
  FpExeclp ('ls',['-l']);
end.
```

32.3.6 FpExecLPE

Synopsis: Execute a program in the path, and pass it an environment

Declaration:
```
function FpExecLPE(const PathName: AnsiString;
                   const S: Array of AnsiString;env: ppchar) : cint
```

Visibility: default

Description: `FpExecLPE` does the same as FpExecLP (1558), but additionally it specifies the environment for the new process in `env`, a pointer to a null-terminated array of null-terminated strings.

Errors: On success, this function does not return.

See also: FpExecLP (1558), FpExecLE (1557)

32.3.7 FpExecV

Synopsis: Execute process

Declaration:
```
function FpExecV(const PathName: AnsiString;args: ppchar) : cint
```

Visibility: default

Description: `FpExecV` replaces the currently running program with the program, specified in `PathName`. 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, `FpExecV` does not return.

Errors: Extended error information is returned by the FpGetErrno (145) function:

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: #rtl.baseunix.fpexecve (139), FpExecvp (1560), FpExecle (1557), FpExecl (1556), FpExeclp (1558), #rtl.baseunix.FpFork (142)

Listing: ./unixex/ex8.pp

```
Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix, 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 }
  fpExecv ('/bin/ls',pp);
end.
```

32.3.8 FpExecVP

Synopsis: Execute process, search path

Declaration: `function FpExecVP(const PathName: AnsiString;args: ppchar) : cint`

Visibility: default

Description: `FpExecVP` replaces the currently running program with the program, specified in `PathName`. 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: Extended error information is returned by the FpGetErrno (145) function:

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: #rtl.baseunix.fpexecve (139), FpExecv (1559), FpExecle (1557), FpExecl (1556), FpExeclp (1558), #rtl.baseunix.FpFork (142)

Listing: ./unixex/ex79.pp

```
Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
      Arg1 : Pchar = '-l';

Var PP : PPchar;
```

```
begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[2]:=Nil;
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable.}
  fpExecvp ('ls',pp);
end.
```

32.3.9 FpExecVPE

Synopsis: Execute process, search path using environment

Declaration:
```
function FpExecVPE(const PathName: AnsiString;args: ppchar;env: ppchar)
                        : cint
```

Visibility: default

Description: `FpExecVP` replaces the currently running program with the program, specified in `PathName`. 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 environment in `Env` is passed to the program. On success, `execvp` does not return.

Errors: Extended error information is returned by the FpGetErrno (145) function:

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

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: #rtl.baseunix.fpexecve (139), FpExecv (1559), FpExecle (1557), FpExecl (1556), FpExeclp (1558), #rtl.baseunix.FpFork (142)

Listing: ./unixex/ex79.pp

```
Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
      Arg1 : Pchar = '-l';

Var PP : PPchar;
```

```
begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[2]:=Nil;
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable.}
  fpExecvp ('ls',pp);
end.
```

32.3.10 fpFlock

Synopsis: Lock a file (advisory lock)

Declaration:
```
function fpFlock(var T: text;mode: cint) : cint
function fpFlock(var F: File;mode: cint) : cint
function fpFlock(fd: cint;mode: cint) : cint
```

Visibility: default

Description: `FpFLock` 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_SHsets a shared lock.

LOCK_EXsets an exclusive lock.

LOCK_UNunlocks the file.

LOCK_NB This can be OR-ed together with the other. If this is done the application doesn't block when locking.

The function returns zero if successful, a nonzero return value indicates an error.

Errors: Extended error information is returned by the FpGetErrno (145) function:

See also: #rtl.baseunix.FpFcntl (141), FSync (1566)

32.3.11 fpfStatFS

Synopsis: Retrieve filesystem information.

Declaration:
```
function fpfStatFS(Fd: cint;Info: PStatFS) : cint
```

Visibility: default

Description: `fpStatFS` returns in `Info` information about the filesystem on which the open file descriptor `fd` resides. Info is of type `tstatfs`. The function returns `0` if the call was succesfull, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: fpStatFS (1563), fpfStat (1538)

32.3.12 fpfsync

Synopsis: Flush cached data to disk

Declaration:
```
function fpfsync(fd: cint) : cint
```

Visibility: default

Description: `fpfsync` forces the system to write all paged (in-memory) changes to file descriptor `fd` to disk. If the call was succesful, 0 is returned.

Errors: On error, a nonzero error-code is returned.

32.3.13 fpgettimeofday

Synopsis: Return kernel time of day in GMT

Declaration:
```
function fpgettimeofday(tp: ptimeval;tzp: ptimezone) : cint
```

Visibility: default

Description: `FpGetTimeOfDay` returns the number of seconds since 00:00, January 1 1970, GMT in a `timeval` record. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call.

Errors: None.

32.3.14 fpStatFS

Synopsis: Retrieve filesystem information.

Declaration:
```
function fpStatFS(Path: pchar;Info: PStatFS) : cint
```

Visibility: default

Description: `fpStatFS` returns in `Info` information about the filesystem on which the file or path `Path` resides. Info is of type `tstatfs`. The function returns `0` if the call was succesfull, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: fpFStatFS (1562), fpStat (1538)

32.3.15 fpSystem

Synopsis: Execute and feed command to system shell

Declaration:
```
function fpSystem(const Command: String) : cint
function fpSystem(const Command: AnsiString) : cint
```

Visibility: default

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 FpFork (142) or FpExecve (139) calls.

Errors: Errors are reported in fpErrNo (94)

See also: POpen (1568), Shell (1570), #rtl.baseunix.FpFork (142), #rtl.baseunix.fpexecve (139)

Listing: ./unixex/ex80.pp

```
program example56;

uses Unix;

{ Program to demonstrate the Shell function }

Var S : Longint;

begin
  WriteIn ('Output of ls -l *.pp');
  S:=fpSystem('ls -l *.pp');
  WriteIn ('Command exited wwith status : ',S);
end.
```

32.3.16 FSearch

Synopsis: Search for file in search path.

Declaration:
```
function FSearch(const path: AnsiString;dirlist: Ansistring;
                 CurrentDirStrategy: TFSearchOption) : AnsiString
function FSearch(const path: AnsiString;dirlist: AnsiString)
                 : AnsiString
```

Visibility: default

Description: `FSearch` searches in `DirList`, a colon separated list of directories, for a file named `Path`. It then returns a path to the found file.

The `CurrentDirStrategy` determines how the current directory is treated when searching:

NoCurrentDirectoryDo not search the current directory unless it is specified in the search path.

CurrentDirectoryFirstSearch the current directory first, before all directories in the search path.

CurrentDirectoryLastSearch the current directory last, after all directories in the search path

It is mainly provided to mimic DOS search path behaviour. Default behaviour is to search the current directory first.

Errors: An empty string if no such file was found.

See also: #rtl.unixutil.FNMatch (1592)

Listing: ./unixex/ex46.pp

```
Program Example46;

{ Program to demonstrate the FSearch function. }

Uses BaseUnix, Unix, Strings;

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

32.3.17 fStatFS

Synopsis: Retrieve filesystem information from a file descriptor.

Declaration: `function fStatFS(Fd: cint;var Info: tstatfs) : cint`

Visibility: default

Description: `fStatFS` returns in `Info` information about the filesystem on which the file with file descriptor `fd` resides. `Info` is of type TStatFS (1588).

The function returns zero if the call was succesful, a nonzero value is returned if the call failed.

Errors: Extended error information is returned by the FpGetErrno (145) function:

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: StatFS (1572), #rtl.baseunix.FpLStat (152)

Listing: ./unixex/ex91.pp

```
program Example30;

{ Program to demonstrate the FSStat function. }

uses BaseUnix,Unix,UnixType;

var s : string;
    fd : cint;
    info : tstatfs;

begin
  writeln ('Info about current partition : ');
  s:='.';
  while s<>'q' do
    begin
    Fd:=fpOpen(S,O_RDOnly);
    if (fd>=0) then
      begin
      if fpfstatfs (fd,@info)<>0 then
        begin
        writeln('Fstat failed. Errno : ',fpgeterrno);
        halt (1);
        end;
      FpClose(fd);
      writeln;
      writeln ('Result of fsstat on file ''',s,'''.');
{$if defined(Linux) or defined(sunos)}
      // SysV like.
      writeln ('fstype  : ',info.fstype);
{$else}
      // BSD like, incl Mac OS X.
      writeln ('fstype  : ',info.ftype);
```

```
{$endif}

      writeln ('bsize   : ',info.bsize);
      writeln ('bfree   : ',info.bfree);
      writeln ('bavail  : ',info.bavail);
      writeln ('files   : ',info.files);
      writeln ('ffree   : ',info.ffree);
      {$ifdef FreeBSD}
      writeln ('fsid    : ',info.fsid[0]);
      {$else}
      writeln ('fsid    : ',info.fsid[0]);
      writeln ('Namelen : ',info.namelen);
      {$endif}
      write ('Type name of file to do fsstat. (q quits) :');
      readln (s)

      end;
    end;
end.
```

32.3.18 fsync

Synopsis: Synchronize file's kernel data with disk.

Declaration: `function fsync(fd: cint) : cint`

Visibility: default

Description: `Fsync` synchronizes the kernel data for file `fd` (the cache) with the disk. The call will not return till all file data was written to disk.

If the call was succesfull, 0 is returned. On failure, a nonzero value is returned.

Errors: Extended error information is returned by the FpGetErrno (145) function:

See also: FpFLock (1562)

32.3.19 GetDomainName

Synopsis: Return current domain name

Declaration: `function GetDomainName : String`

Visibility: default

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 (1567)

Listing: ./unixex/ex39.pp

```
Program Example39;

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

```
Uses Unix;

begin
  WriteIn ('Domain name of this machine is : ',GetDomainName);
end.
```

32.3.20 GetHostName

Synopsis: Return host name

Declaration: `function GetHostName : String`

Visibility: default

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 (1566)

Listing: ./unixex/ex40.pp

```
Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses unix;

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

32.3.21 GetLocalTimezone

Synopsis: Return local timzeone information

Declaration:
```
procedure GetLocalTimezone(timer: cint;var leap_correct: cint;
                           var leap_hit: cint)
procedure GetLocalTimezone(timer: cint)
```

Visibility: default

Description: `GetLocalTimeZone` returns the local timezone information. It also initializes the `TZSeconds` variable, which is used to correct the epoch time to local time.

There should never be any need to call this function directly. It is called by the initialization routines of the Linux unit.

See also: GetTimezoneFile (1568), ReadTimezoneFile (1569)

32.3.22 GetTimezoneFile

Synopsis: Return name of timezone information file

Declaration:
```
function GetTimezoneFile : String
```

Visibility: default

Description: `GetTimezoneFile` returns the location of the current timezone file. The location of file is determined as follows:

1. If /etc/timezone exists, it is read, and the contents of this file is returned. This should work on Debian systems.
2. If /usr/lib/zoneinfo/localtime exists, then it is returned. (this file is a symlink to the timezone file on SuSE systems)
3. If /etc/localtime exists, then it is returned. (this file is a symlink to the timezone file on RedHat systems)

Errors: If no file was found, an empty string is returned.

See also: ReadTimezoneFile (1569)

32.3.23 PClose

Synopsis: Close file opened with POpen (1568)

Declaration:
```
function PClose(var F: File) : cint
function PClose(var F: text) : cint
```

Visibility: default

Description: `PClose` closes a file opened with POpen (1568). It waits for the command to complete, and then returns the exit status of the command.

For an example, see POpen (1568)

Errors: Extended error information is returned by the FpGetErrno (145) function.

See also: POpen (1568)

32.3.24 POpen

Synopsis: Pipe file to standard input/output of program

Declaration:
```
function POpen(var F: text;const Prog: Ansistring;rw: Char) : cint
function POpen(var F: File;const Prog: Ansistring;rw: Char) : cint
```

Visibility: default

Description: `POpen` runs the command specified in `Prog`, 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 (1568). The result is the same, but `PClose` returns the exit status of the command `Prog`.

Errors: Extended error information is returned by the FpGetErrno (145) function. Errors are essentially those of the Execve, Dup and AssignPipe commands.

See also: AssignPipe (1554), PClose (1568)

Listing: ./unixex/ex37.pp

```
Program Example37;

{ Program to demonstrate the Popen function. }

uses BaseUnix,Unix;

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);
  fpchmod ('test21a',&755);
  popen (f,'./test21a arg1 arg2','W');
  if fpgeterrno<>0 then
     writeln ('error from POpen : errno : ', fpgeterrno);
  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.
```

32.3.25 ReadTimezoneFile

Synopsis: Read the timezone file and initialize time routines

Declaration: `procedure ReadTimezoneFile(fn: String)`

Visibility: default

Description: `ReadTimeZoneFile` reads the timezone file `fn` and initializes the local time routines based on the information found there.

There should be no need to call this function. The initialization routines of the linux unit call this routine at unit startup.

Errors: None.

See also: GetTimezoneFile (1568), GetLocalTimezone (1567)

32.3.26 SeekDir

Synopsis: Seek to position in directory

Declaration:
```
procedure SeekDir(p: pDir;loc: clong)
```

Visibility: default

Description: `SeekDir` sets the directory pointer to the `loc`-th entry in the directory structure pointed to by `p`.

For an example, see #rtl.baseunix.fpOpenDir (158).

Errors: Extended error information is returned by the FpGetErrno (145) function:

See also: #rtl.baseunix.fpCloseDir (136), #rtl.baseunix.fpReadDir (162), #rtl.baseunix.fpOpenDir (158), TellDir (1573)

32.3.27 SelectText

Synopsis: Wait for event on text file.

Declaration:
```
function SelectText(var T: Text;TimeOut: ptimeval) : cint
function SelectText(var T: Text;TimeOut: cint) : cint
```

Visibility: default

Description: `SelectText` executes the FpSelect (165) 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 #rtl.baseunix.FpSelect (165). `SYS_EBADF` can also mean that the file wasn't opened.

See also: #rtl.baseunix.FpSelect (165)

32.3.28 Shell

Synopsis: Execute and feed command to system shell

Declaration:
```
function Shell(const Command: String) : cint
function Shell(const Command: AnsiString) : cint
```

Visibility: default

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 FpFork (142) or FpExecve (139) calls.

Errors: Extended error information is returned by the FpGetErrno (145) function:

See also: POpen (1568), FpSystem (1563), #rtl.baseunix.FpFork (142), #rtl.baseunix.fpexecve (139)

Listing: ./unixex/ex56.pp

```
program example56;

uses Unix;

{ 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.
```

32.3.29 SigRaise

Synopsis: Raise a signal (send to current process)

Declaration: `procedure SigRaise(sig: Integer)`

Visibility: default

Description: `SigRaise` sends a `Sig` signal to the current process.

Errors: None.

See also: #rtl.baseunix.FpKill (149), #rtl.baseunix.FpGetPid (147)

Listing: ./unixex/ex65.pp

```
Program example64;

{ Program to demonstrate the SigRaise function.}

uses Unix,BaseUnix;

Var
   oa,na : PSigActionrec;

Procedure DoSig(sig : Longint);cdecl;

begin
   writeln('Receiving signal: ',sig);
end;

begin
   new(na);
   new(oa);
   na^.sa_handler:=SigActionHandler(@DoSig);
   fillchar(na^.Sa_Mask,sizeof(na^.Sa_Mask),#0);
   na^.Sa_Flags:=0;
   {$ifdef Linux}
   // this member is linux only, and afaik even there arcane
   na^.Sa_Restorer:=Nil;
   {$endif}
   if fpSigAction(SigUsr1,na,oa)<>0 then
     begin
     writeln('Error: ',fpgeterrno);
     halt(1);
     end;
   Writeln('Sending USR1 (',sigusr1,') signal to self.');
   SigRaise(sigusr1);
end.
```

32.3.30 StatFS

Synopsis: Retrieve filesystem information from a path.

Declaration:
```
function StatFS(Path: pchar;var Info: tstatfs) : cint
function StatFS(Path: ansistring;var Info: tstatfs) : cint
```

Visibility: default

Description: `StatFS` returns in `Info` information about the filesystem on which the file `Path` resides. `Info` is of type TStatFS (1588).

The function returns zero if the call was succesful, a nonzero value is returned if the call failed.

Errors: Extended error information is returned by the FpGetErrno (145) function:

sys_enotdir A 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_eloop A circular symbolic link was encountered in `Path`.

sys_eio An error occurred while reading from the filesystem.

See also: #rtl.baseunix.FpStat (173), #rtl.baseunix.FpLStat (152)

Listing: ./unixex/ex91.pp

```
program Example30;

{ Program to demonstrate the FSStat function. }

uses BaseUnix,Unix,UnixType;

var s : string;
    fd : cint;
    info : tstatfs;

begin
  writeln ('Info about current partition : ');
  s:='.';
  while s<>'q' do
    begin
    Fd:=fpOpen(S,O_RDOnly);
    if (fd>=0) then
      begin
      if fpfstatfs (fd,@info)<>0 then
        begin
        writeln('Fstat failed. Errno : ',fpgeterrno);
        halt (1);
        end;
      FpClose(fd);
      writeln;
      writeln ('Result of fsstat on file ''',s,'''.');
{$if defined(Linux) or defined(sunos)}
      // SysV like.
      writeln ('fstype  : ',info.fstype);
{$else}
      // BSD like, incl Mac OS X.
```

```
      writeln ('fstype  : ',info.ftype);
{$endif}

      writeln ('bsize   : ',info.bsize);
      writeln ('bfree   : ',info.bfree);
      writeln ('bavail  : ',info.bavail);
      writeln ('files   : ',info.files);
      writeln ('ffree   : ',info.ffree);
      {$ifdef FreeBSD}
      writeln ('fsid    : ',info.fsid[0]);
      {$else}
      writeln ('fsid    : ',info.fsid[0]);
      writeln ('Namelen : ',info.namelen);
      {$endif}
      write ('Type name of file to do fsstat. (q quits) :');
      readln (s)

      end;
    end;
end.
```

32.3.31 TellDir

Synopsis: Return current location in a directory

Declaration: `function TellDir(p: pDir) : TOff`

Visibility: default

Description: `TellDir` returns the current location in the directory structure pointed to by `p`. It returns -1 on failure.

For an example, see #rtl.baseunix.fpOpenDir (158).

Errors:

See also: #rtl.baseunix.fpCloseDir (136), #rtl.baseunix.fpReadDir (162), #rtl.baseunix.fpOpenDir (158), SeekDir (1570)

32.3.32 WaitProcess

Synopsis: Wait for process to terminate.

Declaration: `function WaitProcess(Pid: cint) : cint`

Visibility: default

Description: `WaitProcess` waits for process `PID` to exit. `WaitProcess` is equivalent to the #rtl.baseunix.FpWaitPID (181) call:

```
FpWaitPid(PID,@result,0)
```

Handles of Signal interrupts (errno=EINTR), and returns the Exitcode of Process `PID` (>=0) or -Status if it was terminated

Errors: None.

See also: #rtl.baseunix.FpWaitPID (181), #rtl.baseunix.WTERMSIG (184), #rtl.baseunix.WSTOPSIG (183), #rtl.baseunix.WIFEXITED (183), WIFSTOPPED (1574), #rtl.baseunix.WIFSIGNALED (183), W_EXITCODE (1574), W_STOPCODE (1574), #rtl.baseunix.WEXITSTATUS (183)

32.3.33 WIFSTOPPED

Synopsis: Check whether the process is currently stopped.

Declaration: `function WIFSTOPPED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFSTOPPED` checks `Status` and returns `true` if the process is currently stopped. This is only possible if WUNTRACED was specified in the options of FpWaitPID (181).

See also: #rtl.baseunix.FpWaitPID (181), WaitProcess (1573), #rtl.baseunix.WTERMSIG (184), #rtl.baseunix.WSTOPSIG (183), #rtl.baseunix.WIFEXITED (183), #rtl.baseunix.WIFSIGNALED (183), W_EXITCODE (1574), W_STOPCODE (1574), #rtl.baseunix.WEXITSTATUS (183)

32.3.34 W_EXITCODE

Synopsis: Construct an exit status based on an return code and signal.

Declaration: `function W_EXITCODE(ReturnCode: Integer;Signal: Integer) : Integer`

Visibility: default

Description: `W_EXITCODE` combines `ReturnCode` and `Signal` to a status code fit for `WaitPid`.

See also: #rtl.baseunix.FpWaitPID (181), WaitProcess (1573), #rtl.baseunix.WTERMSIG (184), #rtl.baseunix.WSTOPSIG (183), #rtl.baseunix.WIFEXITED (183), WIFSTOPPED (1574), #rtl.baseunix.WIFSIGNALED (183), W_EXITCODE (1574), W_STOPCODE (1574), #rtl.baseunix.WEXITSTATUS (183)

32.3.35 W_STOPCODE

Synopsis: Construct an exit status based on a signal.

Declaration: `function W_STOPCODE(Signal: Integer) : Integer`

Visibility: default

Description: `W_STOPCODE` constructs an exit status based on `Signal`, which will cause WIFSIGNALED (183) to return `True`

See also: #rtl.baseunix.FpWaitPID (181), WaitProcess (1573), #rtl.baseunix.WTERMSIG (184), #rtl.baseunix.WSTOPSIG (183), #rtl.baseunix.WIFEXITED (183), WIFSTOPPED (1574), #rtl.baseunix.WIFSIGNALED (183), W_EXITCODE (1574), #rtl.baseunix.WEXITSTATUS (183)

Chapter 33

Reference for unit 'unixtype'

33.1 Overview

The unixtype unit contains the definitions of basic unix types. It was initially implemented by Marco van de Voort.

When porting to a new unix platform, this unit should be adapted to the sizes and conventions of the platform to which the compiler is ported.

33.2 Constants, types and variables

33.2.1 Constants

```
ARG_MAX = 131072
```

Max number of command-line arguments.

```
NAME_MAX = 255
```

Max length (in bytes) of filename

```
PATH_MAX = 4095
```

Max length (in bytes) of pathname

```
Prio_PGrp = 1
```

rtl.unix.fpGetPriority (1575) option: Get process group priority.

```
Prio_Process = 0
```

#rtl.unix.fpGetPriority (1538) option: Get process priority.

```
Prio_User = 2
```

#rtl.unix.fpGetPriority (1538) option: Get user priority.

```
SIG_MAXSIG = 128
```

Maximum signal number.

```
SYS_NMLN = 65
```

Max system namelength

```
_PTHREAD_MUTEX_ADAPTIVE_NP = 3
```

Mutex options:

```
_PTHREAD_MUTEX_DEFAULT = _PTHREAD_MUTEX_NORMAL
```

Mutex options:

```
_PTHREAD_MUTEX_ERRORCHECK = _PTHREAD_MUTEX_ERRORCHECK_NP
```

Mutex options:

```
_PTHREAD_MUTEX_ERRORCHECK_NP = 2
```

Mutex options: double lock returns an error code.

```
_PTHREAD_MUTEX_FAST_NP = _PTHREAD_MUTEX_ADAPTIVE_NP
```

Mutex options: Fast mutex

```
_PTHREAD_MUTEX_NORMAL = _PTHREAD_MUTEX_TIMED_NP
```

Mutex options:

```
_PTHREAD_MUTEX_RECURSIVE = _PTHREAD_MUTEX_RECURSIVE_NP
```

Mutex options:

```
_PTHREAD_MUTEX_RECURSIVE_NP = 1
```

Mutex options: recursive mutex

```
_PTHREAD_MUTEX_TIMED_NP = 0
```

Mutex options: ?

33.2.2 Types

```
cbool = longbool
```

Boolean type

```
cchar = cint8
```

C type: 8-bit signed integer

```
cdouble = double
```

Double precision real format.

```
cfloat = single
```

Floating-point real format

```
cint = cint32
```

C type: integer (natural size)

```
cint16 = SmallInt
```

C type: 16 bits sized, signed integer.

```
cint32 = LongInt
```

C type: 32 bits sized, signed integer.

```
cint64 = Int64
```

C type: 64 bits sized, signed integer.

```
cint8 = ShortInt
```

C type: 8 bits sized, signed integer.

```
clock_t = culong
```

Clock ticks type

```
clong = LongInt
```

C type: long signed integer (double sized)

```
clongdouble = extended
```

Usually translates to an extended, but is CPU dependent.

```
clonglong = cint64
```

C type: 64-bit (double long) signed integer.

```
cschar = cint8
```

Signed character type

`cshort = cint16`

C type: short signed integer (half sized)

`csigned = cint`

`csigned` is an alias for cint (1577).

`csint = cint32`

Signed integer

`cslong = LongInt`

The size is CPU dependent.

`cslonglong = cint64`

`cslonglong` is an alias for clonglong (1577).

`csshort = cint16`

Short signed integer type

`cuchar = cuint8`

C type: 8-bit unsigned integer

`cuint = cuint32`

C type: unsigned integer (natural size)

`cuint16 = Word`

C type: 16 bits sized, unsigned integer.

`cuint32 = LongWord`

C type: 32 bits sized, unsigned integer.

`cuint64 = qword`

C type: 64 bits sized, unsigned integer.

`cuint8 = Byte`

C type: 8 bits sized, unsigned integer.

`culong = cardinal`

C type: long unsigned integer (double sized)

```
culonglong = cuint64
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = cuint
```

Alias for #rtl.unixtype.cuint (1578)

```
cushort = cuint16
```

C type: short unsigned integer (half sized)

```
dev_t = cuint64
```

Device descriptor type.

```
gid_t = cuint32
```

Group ID type.

```
ino64_t = cuint64
```

`ino64_t` is an inode type capable of containing 64-bit inodes.

```
ino_t = clong
```

Inode type.

```
ipc_pid_t = cushort
```

Process ID

```
kDev_t = cushort
```

Kernel device type

```
mbstate_t = record
  __count : cint;
  __value : mbstate_value_t;
end
```

This type should never be used directly.

```
mbstate_value_t = record
end
```

This type should never be used directly. It is part of the mbstate_t (1579) type.

```
mode_t = cuint32
```

Inode mode type.

```
nlink_t = cuint32
```

Number of links type.

```
off_t = cint
```

Offset type.

```
pcbool = ^cbool
```

Pointer to boolean type cbool (1576)

```
pcchar = ^cchar
```

Pointer to #rtl.UnixType.cchar (1577)

```
pcdouble = ^cdouble
```

Pointer to cdouble (1577) type.

```
pcfloat = ^cfloat
```

Pointer to cfloat (1577) type.

```
pcint = ^cint
```

Pointer to cInt (1577) type.

```
pcint16 = ^cint16
```

Pointer to 16-bit signed integer type

```
pcint32 = ^cint32
```

Pointer to signed 32-bit integer type

```
pcint64 = ^cint64
```

Pointer to signed 64-bit integer type

```
pcint8 = ^cint8
```

Pointer to 8-bits signed integer type

```
pClock = ^clock_t
```

Pointer to TClock (1586) type.

```
pclong = ^clong
```

Pointer to cLong (1577) type.

```
pclongdouble = ^clongdouble
```

Pointer to the long double type clongdouble (1577)

```
pclonglong = ^clonglong
```

Pointer to longlong type.

```
pcschar = ^cschar
```

Pointer to character type cschar (1577).

```
pcshort = ^cshort
```

Pointer to cShort (1578) type.

```
pcsigned = ^csigned
```

Pointer to signed integer type csigned (1578).

```
pcsint = ^csint
```

Pointer to signed integer type csint (1578)

```
pcslong = ^cslong
```

Pointer ot the signed long cslong (1578)

```
pcslonglong = ^cslonglong
```

Pointer to Signed longlong type cslonglong (1578)

```
pcsshort = ^csshort
```

Pointer to short signed integer type csshort (1578)

```
pcuchar = ^cuchar
```

Pointer to #rtl.UnixType.cuchar (1578)

```
pcuint = ^cuint
```

Pointer to cUInt (1578) type.

```
pcuint16 = ^cuint16
```

Pointer to 16-bit unsigned integer type

```
pcuint32 = ^cuint32
```

Pointer to unsigned 32-bit integer type

```
pcuint64 = ^cuint64
```

Pointer to unsigned 64-bit integer type

```
pcuint8 = ^cuint8
```

Pointer to 8-bits unsigned integer type

```
pculong = ^culong
```

Pointer to cuLong (1578) type.

```
pculonglong = ^culonglong
```

Unsigned longlong type

```
pcunsigned = ^cunsigned
```

Pointer to #rtl.unixtype.cunsigned (1579)

```
pcushort = ^cushort
```

Pointer to cuShort (1579) type.

```
pDev = ^dev_t
```

Pointer to TDev (1586) type.

```
pGid = ^gid_t
```

Pointer to TGid (1586) type.

```
pid_t = cint
```

Process ID type.

```
pIno = ^ino_t
```

Pointer to TIno (1587) type.

```
pIno64 = ^ino64_t
```

Pointer to ino64_t (1579)

```
pkDev = ^kDev_t
```

Pointer to TkDev (1587) type.

```
pmbstate_t = ^mbstate_t
```

Pointer to mbstate_t (1575) type

```
pMode = ^mode_t
```

Pointer to TMode (1587) type.

```
pnLink = ^nlink_t
```

Pointer to TnLink (1587) type.

```
pOff = ^off_t
```

Pointer to TOff (1587) type.

```
pPid = ^pid_t
```

Pointer to TPid (1587) type.

```
pSize = ^size_t
```

Pointer to TSize (1587) type.

```
psize_t = pSize
```

Pointer to size_t (1575) type.

```
pSockLen = ^socklen_t
```

Pointer to TSockLen (1587) type.

```
pSSize = ^ssize_t
```

Pointer to TsSize (1587) type

```
PStatFS = ^TStatfs
```

Pointer to TStatFS (1588) type.

```
pthread_attr_t = record
  __detachstate : cint;
  __schedpolicy : cint;
  __schedparam : sched_param;
  __inheritsched : cint;
  __scope : cint;
  __guardsize : size_t;
  __stackaddr_set : cint;
  __stackaddr : pointer;
  __stacksize : size_t;
end
```

pthread_attr_t describes the thread attributes. It should be considered an opaque record, the names of the fields can change anytime. Use the appropriate functions to set the thread attributes.

```
pthread_condattr_t = record
  __dummy : cint;
end
```

pthread_condattr_t describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_cond_t = record
  __c_lock : _pthread_fastlock;
  __c_waiting : pointer;
  __padding : Array[0..48-1-sizeof(_pthread_fastlock)-sizeof(pointer)-sizeof(clonglo
  __align : clonglong;
end
```

pthread_cond_t describes a thread conditional variable. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_key_t = cuint
```

Thread local storage key (opaque)

```
pthread_mutexattr_t = record
  __mutexkind : cint;
end
```

pthread_mutexattr_t describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_mutex_t = record
  __m_reserved : cint;
  __m_count : cint;
  __m_owner : pointer;
  __m_kind : cint;
  __m_lock : _pthread_fastlock;
end
```

_pthread_mutex_t describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_rwlockattr_t = record
  __lockkind : cint;
  __pshared : cint;
end
```

pthread_rwlockattr_t describes the attributes of a lock. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_rwlock_t = record
  __rw_readers : cint;
  __rw_writer : pointer;
  __rw_read_waiting : pointer;
  __rw_write_waiting : pointer;
  __rw_kind : cint;
  __rw_pshared : cint;
end
```

pthread_rwlock_t describes a lock. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_t = culong
```

Thread description record

```
pTime = ^time_t
```

Pointer to TTime (1588) type.

```
ptimespec = ^timespec
```

Pointer to timespec (1586) record.

```
ptimeval = ^timeval
```

Pointer to timeval (1586) record.

```
ptime_t = ^time_t
```

Pointer to time_t (1587) type.

```
pUid = ^uid_t
```

Pointer to TUid (1588) type.

```
pwchar_t = ^wchar_t
```

Pointer to wchar_t (1575) type.

```
sched_param = record
  __sched_priority : cint;
end
```

Scheduling parameter description record.

```
sem_t = record
  __sem_lock : _pthread_fastlock;
  __sem_value : cint;
  __sem_waiting : pointer;
end
```

sem_t describes a thread semaphore. It should be considered an opaque record, the names of the fields can change anytime.

```
size_t = cuint32
```

Size specification type.

```
socklen_t = cuint32
```

Socket address length type.

```
ssize_t = cint32
```

Small size type.

```
TClock = clock_t
```

Alias for clock_t (1577) type.

```
TDev = dev_t
```

Alias for dev_t (1579) type.

```
TGid = gid_t
```

Alias for gid_t (1579) type.

```
timespec = packed record
  tv_sec : time_t;
  tv_nsec : clong;
end
```

Record specifying time interval.

```
timeval = packed record
  tv_sec : time_t;
  tv_usec : clong;
end
```

Time specification type.

```
time_t = clong
```

Time span type

```
TIno = ino_t
```

Alias for ino_t (1579) type.

```
TIno64 = ino64_t
```

Alias for ino64_t (1579)

```
TIOCtlRequest = cint
```

Opaque type used in FpIOCtl (149)

```
TkDev = kDev_t
```

Alias for kDev_t (1579) type.

```
TMode = mode_t
```

Alias for mode_t (1580) type.

```
TnLink = nlink_t
```

Alias for nlink_t (1580) type.

```
TOff = off_t
```

Alias for off_t (1580) type.

```
TPid = pid_t
```

Alias for pid_t (1582) type.

```
TSize = size_t
```

Alias for size_t (1586) type

```
TSockLen = socklen_t
```

Alias for socklen_t (1586) type.

```
TSSize = ssize_t
```

Alias for ssize_t (1586) type

```
TStatfs = packed record
  fstype : cint;
  bsize : cint;
  blocks : clong;
  bfree : clong;
```

```
  bavail : clong;
  files : clong;
  ffree : clong;
  fsid : Array[0..1] of cint;
  namelen : clong;
  spare : Array[0..5] of clong;
end
```

Record describing a file system in the baseunix.fpstatfs (1575) call.

```
TTime = time_t
```

Alias for TTime (1588) type.

```
TTimeSpec = timespec
```

Alias for TimeSpec (1586) type.

```
TTimeVal = timeval
```

Alias for TimeVal (1586) record.

```
TUid = uid_t
```

Alias for uid_t (1588) type.

```
uid_t = cuint32
```

User ID type

```
wchar_t = cint32
```

Wide character type.

```
wint_t = cint32
```

Wide character size type.

```
_pthread_fastlock = record
  __status : clong;
  __spinlock : cint;
end
```

`_pthread_fastlock` describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

Chapter 34

Reference for unit 'unixutil'

34.1 Overview

The UnixUtil unit contains some of the routines that were present in the old Linux unit, but which do not really belong in the unix (1538) or baseunix (94) units.

Most of the functions described here have cross-platform counterparts in the SysUtils (1350) unit. It is therefore recommended to use that unit.

34.2 Constants, types and variables

34.2.1 Types

`ComStr =`

Command-line string type.

`DirStr =`

Filename directory part string type.

`ExtStr =`

Filename extension part string type.

`NameStr =`

Filename name part string type.

`PathStr =`

Filename full path string type.

34.2.2 Variables

`Tzseconds : LongInt`

Seconds west of GMT

34.3 Procedures and functions

34.3.1 ArrayStringToPPchar

Synopsis: Concert an array of string to an array of null-terminated strings

Declaration:
```
function ArrayStringToPPchar(const S: Array of AnsiString;
                             reserveentries: LongInt) : ppchar
```

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: StringToPPChar (1595)

34.3.2 Basename

Synopsis: Return basename of a file

Declaration:
```
function Basename(const path: PathStr;const suf: PathStr) : PathStr
```

Visibility: default

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 (1591)

Listing: ./unutilex/ex48.pp

```
Program Example48;

{ Program to demonstrate the BaseName function. }

Uses Dos, Unix, UnixUtil;

Var S : String;

begin
  S:=FExpand(Paramstr(0));
  Writeln ('This program is called : ',Basename(S,''));
end.
```

34.3.3 Dirname

Synopsis: Extract directory part from filename

Declaration:
```
function Dirname(const path: PathStr) : PathStr
```

Visibility: default

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 (1590)

Listing: ./unutilex/ex47.pp

```
Program Example47;

{ Program to demonstrate the DirName function. }

Uses Dos, Unix, UnixUtil;

Var S : String;

begin
  S:=FExpand(Paramstr(0));
  WriteIn ('This program is in directory : ',Dirname(S));
end.
```

34.3.4 EpochToLocal

Synopsis: Convert epoch time to local time

Declaration:
```
procedure EpochToLocal(epoch: LongInt;var year: Word;var month: Word;
                       var day: Word;var hour: Word;var minute: Word;
                       var second: Word)
```

Visibility: default

Description: Converts the epoch time (=Number of seconds since 00:00:00 , January 1, 1970, corrected for your time zone) to local date and time.

This function takes into account the timzeone settings of your system.

Errors: None

See also: LocalToEpoch (1594)

Listing: ./unutilex/ex3.pp

```
Program Example3;

{ Program to demonstrate the EpochToLocal function. }

Uses BaseUnix, Unix, UnixUtil;

Var Year, month, day, hour, minute, seconds : Word;
```

```
begin
  EpochToLocal (FPTime,Year,month,day,hour,minute,seconds);
  WriteIn ('Current date : ',Day:2,'/',Month:2,'/',Year:4);
  WriteIn ('Current time : ',Hour:2,':',minute:2,':',seconds:2);
end.
```

34.3.5 FNMatch

Synopsis: Check whether filename matches wildcard specification

Declaration: `function FNMatch(const Pattern: String;const Name: String) : Boolean`

Visibility: default

Description: `FNMatch` returns `True` if the filename in `Name` matches the wildcard pattern in `Pattern`, `False` otherwise.

`Pattern` can contain the wildcards * (match zero or more arbitrary characters) or `?` (match a single character).

Errors: None.

See also: #rtl.unix.FSearch (1564)

Listing: ./unutilex/ex69.pp

```
Program Example69;

{ Program to demonstrate the FNMatch function. }

Uses unixutil;

  Procedure TestMatch(Pattern,Name : String);

  begin
    Write ('"',Name,'" ');
    If FNMatch (Pattern,Name) then
      Write ('matches')
    else
      Write ('does not match');
    WriteIn(' "',Pattern,'".');
  end;

begin
  TestMatch('*','FileName');
  TestMatch('.*','FileName');
  TestMatch('*a*','FileName');
  TestMatch('?ile*','FileName');
  TestMatch('?','FileName');
  TestMatch('.?','FileName');
  TestMatch('?a*','FileName');
  TestMatch('??*me?','FileName');
end.
```

34.3.6 FSplit

Synopsis: Split filename into path, name and extension

Declaration:
```
procedure FSplit(const Path: PathStr;var Dir: DirStr;var Name: NameStr;
                 var Ext: ExtStr)
```

Visibility: default

Description: `FSplit` splits a full file name into 3 parts : A `Path`, a `Name` and an extension (in `ext`). The extension is taken to be all letters after the last dot (.).

Errors: None.

See also: #rtl.unix.FSearch (1564)

Listing: ./unutilex/ex67.pp

```
Program Example67;

uses UnixUtil;

{ Program to demonstrate the FSplit function. }

var
  Path,Name,Ext : string;

begin
  FSplit(ParamStr(1),Path,Name,Ext);
  WriteLn('Split ',ParamStr(1),' in:');
  WriteLn('Path      : ',Path);
  WriteLn('Name      : ',Name);
  WriteLn('Extension : ',Ext);
end.
```

34.3.7 GetFS

Synopsis: Return file selector

Declaration:
```
function GetFS(var T: Text) : LongInt
function GetFS(var F: File) : LongInt
```

Visibility: default

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 #rtl.baseunix.fpSelect (165) call or so.

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

See also: #rtl.baseunix.fpSelect (165)

Listing: ./unutilex/ex34.pp

```
Program Example33;

{ Program to demonstrate the SelectText function. }

Uses Unix;
```

```
Var tv : TimeVal;

begin
  WriteIn ('Press the <ENTER> to continue the program.');
  { Wait until File descriptor 0 (=Input) changes }
  SelectText (Input,nil);
  { Get rid of <ENTER> in buffer }
  readln;
  WriteIn ('Press <ENTER> key in less than 2 seconds...');
  tv.tv_sec:=2;
  tv.tv_sec:=0;
  if SelectText (Input,@tv)>0 then
    WriteIn ('Thank you !')
  else
    WriteIn ('Too late !');
end.
```

34.3.8 GregorianToJulian

Synopsis: Converts a gregorian date to a julian date

Declaration:
```
function GregorianToJulian(Year: LongInt;Month: LongInt;Day: LongInt)
                                  : LongInt
```

Visibility: default

Description: `GregorianToJulian` takes a gregorian date and converts it to a Julian day.

Errors: None.

See also: JulianToGregorian (1594)

34.3.9 JulianToGregorian

Synopsis: Converts a julian date to a gregorian date

Declaration:
```
procedure JulianToGregorian(JulianDN: LongInt;var Year: Word;
                                  var Month: Word;var Day: Word)
```

Visibility: default

Description: `JulianToGregorian` takes a julian day and converts it to a gregorian date. (Start of the Julian Date count is from 0 at 12 noon 1 JAN -4712 (4713 BC),)

Errors: None.

See also: GregorianToJulian (1594)

34.3.10 LocalToEpoch

Synopsis: Convert local time to epoch (unix) time

Declaration:
```
function LocalToEpoch(year: Word;month: Word;day: Word;hour: Word;
                           minute: Word;second: Word) : LongInt
```

Visibility: default

Description: Converts the Local time to epoch time (=Number of seconds since 00:00:00 , January 1, 1970).

Errors: None

See also: EpochToLocal (1591)

Listing: ./unutilex/ex4.pp

```
Program Example4;

{ Program to demonstrate the LocalToEpoch function. }

Uses UnixUtil;

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));
  WriteIn (' seconds past 00:00 1/1/1980');
end.
```

34.3.11 StringToPPChar

Synopsis: Split string in list of null-terminated strings

Declaration:
```
function StringToPPChar(S: PChar;ReserveEntries: Integer) : ppchar
function StringToPPChar(var S: String;ReserveEntries: Integer) : ppchar
function StringToPPChar(var S: AnsiString;ReserveEntries: Integer)
                        : ppchar
```

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of pchars that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various `Exec` calls.

Errors: None.

See also: ArrayStringToPPchar (1590), #rtl.baseunix.FpExecve (139)

Listing: ./unutilex/ex70.pp

```
Program Example70;

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

```
Uses UnixUtil;

Var S : String;
    P : PPChar;
    I : longint;

begin
  // remark whitespace at end.
  S:='This is a string with words. ';
  P:=StringToPPChar(S,0);
  I:=0;
  While P[i]<>Nil do
    begin
    WriteIn('Word ',i,' : ',P[i]);
    Inc(I);
    end;
  FreeMem(P,i*SizeOf(Pchar));
end.
```

Chapter 35

Reference for unit 'video'

35.1 Examples utility unit

The examples in this section make use of the unit vidutil, which contains the `TextOut` function. This function writes a text to the screen at a given location. It looks as follows:

35.2 Writing a custom video driver

Writing a custom video driver is not difficult, and generally means implementing a couple of functions, which whould be registered with the SetVideoDriver (1615) function. The various functions that can be implemented are located in the TVideoDriver (1604) record:

```
TVideoDriver = Record
  InitDriver        : Procedure;
  DoneDriver        : Procedure;
  UpdateScreen      : Procedure(Force : Boolean);
  ClearScreen       : Procedure;
  SetVideoMode      : Function (Const Mode : TVideoMode) : Boolean;
  GetVideoModeCount : Function : Word;
  GetVideoModeData  : Function(Index : Word; Var Data : TVideoMode) : Boolean;
  SetCursorPos      : procedure (NewCursorX, NewCursorY: Word);
  GetCursorType     : function : Word;
  SetCursorType     : procedure (NewType: Word);
  GetCapabilities   : Function : Word;
end;
```

Not all of these functions must be implemented. In fact, the only absolutely necessary function to write a functioning driver is the `UpdateScreen` function. The general calls in the Video unit will check which functionality is implemented by the driver.

The functionality of these calls is the same as the functionality of the calls in the video unit, so the expected behaviour can be found in the previous section. Some of the calls, however, need some additional remarks.

InitDriver Called by `InitVideo`, this function should initialize any data structures needed for the functionality of the driver, maybe do some screen initializations. The function is guaranteed to be called only once; It can only be called again after a call to `DoneVideo`. The variables

ScreenWidth and ScreenHeight should be initialized correctly after a call to this function, as the InitVideo call will initialize the VideoBuf and OldVideoBuf arrays based on their values.

DoneDriver This should clean up any structures that have been initialized in the InitDriver function. It should possibly also restore the screen as it was before the driver was initialized. The VideoBuf and OldVideoBuf arrays will be disposed of by the general DoneVideo call.

UpdateScreen This is the only required function of the driver. It should update the screen based on the VideoBuf array's contents. It can optimize this process by comparing the values with values in the OldVideoBuf array. After updating the screen, the UpdateScreen procedure should update the OldVideoBuf by itself. If the Force parameter is True, the whole screen should be updated, not just the changed values.

ClearScreen If there is a faster way to clear the screen than to write spaces in all character cells, then it can be implemented here. If the driver does not implement this function, then the general routines will write spaces in all video cells, and will call UpdateScreen(True).

SetVideoMode Should set the desired video mode, if available. It should return True if the mode was set, False if not.

GetVideoModeCount Should return the number of supported video modes. If no modes are supported, this function should not be implemented; the general routines will return 1. (for the current mode)

GetVideoModeData Should return the data for the Index-th mode; Index is zero based. The function should return true if the data was returned correctly, false if Index contains an invalid index. If this is not implemented, then the general routine will return the current video mode when Index equals 0.

GetCapabilities If this function is not implemented, zero (i.e. no capabilities) will be returned by the general function.

The following unit shows how to override a video driver, with a driver that writes debug information to a file. The unit can be used in any of the demonstration programs, by simply including it in the uses clause. Setting DetailedVideoLogging to True will create a more detailed log (but will also slow down functioning)

35.3 Overview

The Video unit implements a screen access layer which is system independent. It can be used to write on the screen in a system-independent way, which should be optimal on all platforms for which the unit is implemented.

The working of the Video is simple: After calling InitVideo (1612), the array VideoBuf contains a representation of the video screen of size ScreenWidth*ScreenHeight, going from left to right and top to bottom when walking the array elements: VideoBuf[0] contains the character and color code of the top-left character on the screen. VideoBuf[ScreenWidth] contains the data for the character in the first column of the second row on the screen, and so on.

To write to the 'screen', the text to be written should be written to the VideoBuf array. Calling UpdateScreen (1616) will then cp the text to the screen in the most optimal way. (an example can be found further on).

The color attribute is a combination of the foreground and background color, plus the blink bit. The bits describe the various color combinations:

bits 0-3 The foreground color. Can be set using all color constants.

bits 4-6 The background color. Can be set using a subset of the color constants.

bit 7 The blinking bit. If this bit is set, the character will appear blinking.

Each possible color has a constant associated with it, see the constants section for a list of constants.

The foreground and background color can be combined to a color attribute with the following code:

```
Attr:=ForeGroundColor + (BackGroundColor shl 4);
```

The color attribute can be logically or-ed with the blink attribute to produce a blinking character:

```
Atrr:=Attr or blink;
```

But not all drivers may support this.

The contents of the `VideoBuf` array may be modified: This is 'writing' to the screen. As soon as everything that needs to be written in the array is in the `VideoBuf` array, calling `UpdateScreen` will copy the contents of the array screen to the screen, in a manner that is as efficient as possible.

The updating of the screen can be prohibited to optimize performance; To this end, the LockScreenUpdate (1613) function can be used: This will increment an internal counter. As long as the counter differs from zero, calling UpdateScreen (1616) will not do anything. The counter can be lowered with UnlockScreenUpdate (1615). When it reaches zero, the next call to UpdateScreen (1616) will actually update the screen. This is useful when having nested procedures that do a lot of screen writing.

The video unit also presents an interface for custom screen drivers, thus it is possible to override the default screen driver with a custom screen driver, see the SetVideoDriver (1615) call. The current video driver can be retrieved using the GetVideoDriver (1610) call.

Remark: The video unit should *not* be used together with the crt unit. Doing so will result in very strange behaviour, possibly program crashes.

35.4 Constants, types and variables

35.4.1 Constants

```
Black = 0
```

Black color attribute

```
Blink = 128
```

Blink attribute

```
Blue = 1
```

Blue color attribute

```
Brown = 6
```

Brown color attribute

```
cpBlink = $0002
```

Video driver supports blink attribute

```
cpChangeCursor = $0020
```

Video driver supports changing cursor shape.

```
cpChangeFont = $0008
```

Video driver supports changing screen font.

```
cpChangeMode = $0010
```

Video driver supports changing mode

```
cpColor = $0004
```

Video driver supports color

```
cpUnderLine = $0001
```

Video driver supports underline attribute

```
crBlock = 2
```

Block cursor

```
crHalfBlock = 3
```

Half block cursor

```
crHidden = 0
```

Hide cursor

```
crUnderLine = 1
```

Underline cursor

```
Cyan = 3
```

Cyan color attribute

```
DarkGray = 8
```

Dark gray color attribute

```
errOk = 0
```

No error

```
ErrorCode : LongInt = ErrOK
```

Error code returned by the last operation.

```
ErrorHandler : TErrorHandler = @DefaultErrorHandler
```

The `ErrorHandler` variable can be set to a custom-error handling function. It is set by default to the DefaultErrorHandler (1607) function.

```
ErrorInfo : Pointer = nil
```

Pointer to extended error information.

```
errVioBase = 1000
```

Base value for video errors

```
errVioInit = errVioBase + 1
```

Video driver initialization error.

```
errVioNoSuchMode = errVioBase + 3
```

Invalid video mode

```
errVioNotSupported = errVioBase + 2
```

Unsupported video function

```
FVMaxWidth = 132
```

Maximum screen buffer width.

```
Green = 2
```

Green color attribute

```
iso_codepages = [iso01,iso02,iso03,iso04,iso05,iso06,iso07,iso08,iso09,iso10,iso13,i
```

`iso_codepages` is a set containing all code pages that use an ISO encoding.

```
LightBlue = 9
```

Light Blue color attribute

```
LightCyan = 11
```

Light cyan color attribute

```
LightGray = 7
```

Light gray color attribute

```
LightGreen = 10
```

Light green color attribute

```
LightMagenta = 13
```

Light magenta color attribute

```
LightRed = 12
```

Light red color attribute

```
LowAscii = true
```

On some systems, the low 32 values of the DOS code page are necessary for the ASCII control codes and cannot be displayed by programs. If LowAscii is true, you can use the low 32 ASCII values. If it is false, you must avoid using them.

LowAscii can be implemented either through a constant, variable or property. You should under no circumstances assume that you can write to LowAscii, or take its address.

```
Magenta = 5
```

Magenta color attribute

```
NoExtendedFrame = false
```

The VT100 character set only has line drawing characters consisting of a single line. If this value is true, the line drawing characters with two lines will be automatically converted to single lines.

NoExtendedFrame can be implemented either through a constant, variable or property. You should under no circumstances assume that you can write to NoExtendedFrame, or take its address.

```
Red = 4
```

Red color attribute

```
ScreenHeight : Word = 0
```

Current screen height

```
ScreenWidth : Word = 0
```

Current screen Width

```
vga_codepages = [cp437,cp850,cp852,cp866]
```

`vga_codepages` is a set containing all code pages that can be considered a normal vga font (as in use on early VGA cards) Note that KOI8-R has line drawing characters in wrong place.

```
vioOK = 0
```

No errors occurred

```
White = 15
```

White color attribute

```
Yellow = 14
```

Yellow color attribute

35.4.2 Types

```
PVideoBuf = ^TVideoBuf
```

Pointer type to TVideoBuf (1603)

```
PVideoCell = ^TVideoCell
```

Pointer type to TVideoCell (1604)

```
PVideoMode = ^TVideoMode
```

Pointer to TVideoMode (1605) record.

```
Tencoding = (cp437,cp850,cp852,cp866,koi8r,iso01,iso02,iso03,iso04,
             iso05,iso06,iso07,iso08,iso09,iso10,iso13,iso14,iso15)
```

This type is available under Unix-like operating systems only.

```
TErrorHandler = function(Code: LongInt;Info: Pointer)
                         : TErrorHandlerReturnValue
```

The `TErrorHandler` function is used to register an own error handling function. It should be used when installing a custom error handling function, and must return one of the above values.

`Code` should contain the error code for the error condition, and the `Info` parameter may contain any data type specific to the error code passed to the function.

```
TErrorHandlerReturnValue = (errRetry,errAbort,errContinue)
```

Type used to report and respond to error conditions

```
TVideoBuf = Array[0..32759] of TVideoCell
```

The `TVideoBuf` type represents the screen.

```
TVideoCell = Word
```

Table 35.1: Enumeration values for type Tencoding

Value	Explanation
cp437	Codepage 437
cp850	Codepage 850
cp852	Codepage 852
cp866	Codepage 866
iso01	ISO 8859-1
iso02	ISO 8859-2
iso03	ISO 8859-3
iso04	ISO 8859-4
iso05	ISO 8859-5
iso06	ISO 8859-6
iso07	ISO 8859-7
iso08	ISO 8859-8
iso09	ISO 8859-9
iso10	ISO 8859-10
iso13	ISO 8859-13
iso14	ISO 8859-14
iso15	ISO 8859-15
koi8r	KOI8-R codepage

Table 35.2: Enumeration values for type TErrorHandlerReturnValue

Value	Explanation
errAbort	abort and return error code
errContinue	abort without returning an errorcode.
errRetry	retry the operation

`TVideoCell` describes one character on the screen. One of the bytes contains the color attribute with which the character is drawn on the screen, and the other byte contains the ASCII code of the character to be drawn. The exact position of the different bytes in the record is operating system specific. On most little-endian systems, the high byte represents the color attribute, while the low-byte represents the ASCII code of the character to be drawn.

```
TVideoDriver = record
  InitDriver : procedure;
  DoneDriver : procedure;
  UpdateScreen : procedure(Force: Boolean);
  ClearScreen : procedure;
  SetVideoMode : function(const Mode: TVideoMode) : Boolean;
  GetVideoModeCount : function : Word;
  GetVideoModeData : function(Index: Word;var Data: TVideoMode) : Boolean;
  SetCursorPos : procedure(NewCursorX: Word;NewCursorY: Word);
  GetCursorType : function : Word;
  SetCursorType : procedure(NewType: Word);
  GetCapabilities : function : Word;
end
```

TVideoDriver record can be used to install a custom video driver, with the SetVideoDriver (1615) call.

An explanation of all fields can be found there.

```
TVideoMode = record
  Col : Word;
  Row : Word;
  Color : Boolean;
end
```

The TVideoMode record describes a videomode. Its fields are self-explaining: Col,Row describe the number of columns and rows on the screen for this mode. Color is True if this mode supports colors, or False if not.

```
TVideoModeSelector = function(const VideoMode: TVideoMode;
                              Params: LongInt) : Boolean
```

Video mode selection callback prototype.

35.4.3 Variables

```
CursorLines : Byte
```

CursorLines is a bitmask which determines which cursor lines are visible and which are not. Each set bit corresponds to a cursorline being shown.

This variable is not supported on all platforms, so it should be used sparingly.

```
CursorX : Word
```

Current horizontal position in the screen where items will be written.

```
CursorY : Word
```

Current vertical position in the screen where items will be written.

```
external_codepage : Tencoding
```

This variable is for internal use only and should not be used.

```
internal_codepage : Tencoding
```

This variable is for internal use only and should not be used.

```
OldVideoBuf : PVideoBuf
```

The OldVideoBuf contains the state of the video screen after the last screen update. The UpdateScreen (1616) function uses this array to decide which characters on screen should be updated, and which not.

Note that the OldVideoBuf array may be ignored by some drivers, so it should not be used. The Array is in the interface section of the video unit mainly so drivers that need it can make use of it.

`ScreenColor : Boolean`

`ScreenColor` indicates whether the current screen supports colors.

`VideoBuf : PVideoBuf`

`VideoBuf` forms the heart of the Video unit: This variable represents the physical screen. Writing to this array and calling UpdateScreen (1616) will write the actual characters to the screen.

`VideoBufSize : LongInt`

Current size of the video buffer pointed to by VideoBuf (1606)

35.5 Procedures and functions

35.5.1 ClearScreen

Synopsis: Clear the video screen.

Declaration: `procedure ClearScreen`

Visibility: default

Description: `ClearScreen` clears the entire screen, and calls UpdateScreen (1616) after that. This is done by writing spaces to all character cells of the video buffer in the default color (lightgray on black, color attribute \$07).

Errors: None.

See also: InitVideo (1612), UpdateScreen (1616)

Listing: ./videoex/ex3.pp

```
program testvideo;

uses video,keyboard,vidutil;

Var
  i : longint;
  k : TkeyEvent;

begin
  InitVideo;
  InitKeyboard;
  For I:=1 to 10 do
    TextOut(i,i, 'Press any key to clear screen');
  UpdateScreen(false);
  K:=GetKeyEvent;
  ClearScreen;
  TextOut(1,1,'Cleared screen. Press any key to end');
  UpdateScreen(true);
  K:=GetKeyEvent;
  DoneKeyBoard;
  DoneVideo;
end.
```

35.5.2 DefaultErrorHandler

Synopsis: Default error handling routine.

Declaration:
```
function DefaultErrorHandler(AErrorCode: LongInt;AErrorInfo: Pointer)
                                  : TErrorHandlerReturnValue
```

Visibility: default

Description: `DefaultErrorHandler` is the default error handler used by the video driver. It simply sets the error code `AErrorCode` and `AErrorInfo` in the global variables `ErrorCode` and `ErrorInfo` and returns `errContinue`.

Errors: None.

35.5.3 DoneVideo

Synopsis: Disable video driver.

Declaration:
```
procedure DoneVideo
```

Visibility: default

Description: `DoneVideo` disables the Video driver if the video driver is active. If the videodriver was already disabled or not yet initialized, it does nothing. Disabling the driver means it will clean up any allocated resources, possibly restore the screen in the state it was before `InitVideo` was called. Particularly, the `VideoBuf` and `OldVideoBuf` arrays are no longer valid after a call to `DoneVideo`.

The `DoneVideo` should always be called if `InitVideo` was called. Failing to do so may leave the screen in an unusable state after the program exits.

For an example, see most other functions.

Errors: Normally none. If the driver reports an error, this is done through the `ErrorCode` variable.

See also: InitVideo (1612)

35.5.4 GetCapabilities

Synopsis: Get current driver capabilities.

Declaration:
```
function GetCapabilities : Word
```

Visibility: default

Description: `GetCapabilities` returns the capabilities of the current driver. It is an or-ed combination of the following constants:

cpUnderLineVideo driver supports underline attribute

cpBlinkVideo driver supports blink attribute

cpColorVideo driver supports color

cpChangeFontVideo driver supports changing screen font.

cpChangeModeVideo driver supports changing mode

cpChangeCursorVideo driver supports changing cursor shape.

Note that the video driver should not yet be initialized to use this function. It is a property of the driver.

Errors: None.

See also: GetCursorType (1608), GetVideoDriver (1610)

Listing: ./videoex/ex4.pp

```
Program Example4;

{ Program to demonstrate the GetCapabilities function. }

Uses video;

Var
  W: Word;

  Procedure TestCap(Cap: Word; Msg : String);

  begin
    Write(Msg,' : ');
    If (W and Cap=Cap) then
      Writeln('Yes')
    else
      Writeln('No');
  end;

begin
  W:=GetCapabilities;
  Writeln('Video driver supports following functionality');
  TestCap(cpUnderLine,'Underlined characters');
  TestCap(cpBlink,'Blinking characters');
  TestCap(cpColor,'Color characters');
  TestCap(cpChangeFont,'Changing font');
  TestCap(cpChangeMode,'Changing video mode');
  TestCap(cpChangeCursor,'Changing cursor shape');
end.
```

35.5.5 GetCursorType

Synopsis: Get screen cursor type

Declaration: `function GetCursorType : Word`

Visibility: default

Description: `GetCursorType` returns the current cursor type. It is one of the following values:

crHidden Hide cursor

crUnderLine Underline cursor

crBlock Block cursor

crHalfBlock Half block cursor

Note that not all drivers support all types of cursors.

Errors: None.

See also: SetCursorType (1614), GetCapabilities (1607)

Listing: ./videoex/ex5.pp

```
Program Example5;

{ Program to demonstrate the GetCursorType function. }

Uses video,keyboard,vidutil;

Const
  Cursortypes : Array[crHidden..crHalfBlock] of string =
    ('Hidden','UnderLine','Block','HalfBlock');

begin
  InitVideo;
  InitKeyboard;
  TextOut(1,1,'Cursor type: '+CursorTypes[GetCursorType]);
  TextOut(1,2,'Press any key to exit.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

35.5.6 GetLockScreenCount

Synopsis: Get the screen lock update count.

Declaration: `function GetLockScreenCount : Integer`

Visibility: default

Description: `GetLockScreenCount` returns the current lock level. When the lock level is zero, a call to UpdateScreen (1616) will actually update the screen.

Errors: None.

See also: LockScreenUpdate (1613), UnlockScreenUpdate (1615), UpdateScreen (1616)

Listing: ./videoex/ex6.pp

```
Program Example6;

{ Program to demonstrate the GetLockScreenCount function. }

Uses video,keyboard,vidutil;

Var
  I : Longint;
  S : String;

begin
  InitVideo;
  InitKeyboard;
  TextOut(1,1,'Press key till new text appears.');
  UpdateScreen(False);
  Randomize;
  For I:=0 to Random(10)+1 do
    LockScreenUpdate;
```

```
  I:=0;
  While GetLockScreenCount<>0 do
    begin
    Inc(I);
    Str(I,S);
    UnlockScreenUpdate;
    GetKeyEvent;
    TextOut(1,1,'UnLockScreenUpdate had to be called '+S+' times');
    UpdateScreen(False);
    end;
  TextOut(1,2,'Press any key to end.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

35.5.7 GetVideoDriver

Synopsis: Get a copy of the current video driver.

Declaration: `procedure GetVideoDriver(var Driver: TVideoDriver)`

Visibility: default

Description: `GetVideoMode` returns the settings of the currently active video mode. The `row,col` fields indicate the dimensions of the current video mode, and `Color` is true if the current video supports colors.

Errors: None.

See also: SetVideoMode (1615), GetVideoModeData (1612)

Listing: ./videoex/ex7.pp

```
Program Example7;

{ Program to demonstrate the GetVideoMode function. }

Uses video,keyboard,vidutil;

Var
  M : TVideoMode;
  S : String;

begin
  InitVideo;
  InitKeyboard;
  GetVideoMode(M);
  if M.Color then
    TextOut(1,1,'Current mode has color')
  else
    TextOut(1,1,'Current mode does not have color');
  Str(M.Row,S);
  TextOut(1,2,'Number of rows    : '+S);
  Str(M.Col,S);
  TextOut(1,3,'Number of columns : '+S);
  Textout(1,4,'Press any key to exit.');
```

```
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

35.5.8 GetVideoMode

Synopsis: Return current video mode

Declaration: `procedure GetVideoMode(var Mode: TVideoMode)`

Visibility: default

Description: Return current video mode

35.5.9 GetVideoModeCount

Synopsis: Get the number of video modes supported by the driver.

Declaration: `function GetVideoModeCount : Word`

Visibility: default

Description: `GetVideoModeCount` returns the number of video modes that the current driver supports. If the driver does not support switching of modes, then 1 is returned.

This function can be used in conjunction with the GetVideoModeData (1612) function to retrieve data for the supported video modes.

Errors: None.

See also: GetVideoModeData (1612), GetVideoMode (1611)

Listing: ./videoex/ex8.pp

```
Program Example8;

{ Program to demonstrate the GetVideoModeCount function. }

Uses video,keyboard,vidutil;

Procedure DumpMode (M : TVideoMode; Index : Integer);

Var
 S : String;

begin
  Str(Index:2,S);
  inc(Index);
  TextOut(1,Index,'Data for mode '+S+': ');
  if M.Color then
    TextOut(19,Index,'    color,')
  else
    TextOut(19,Index,'No color,');
  Str(M.Row:3,S);
  TextOut(28,Index,S+' rows');
  Str(M.Col:3,S);
```

```
  TextOut(36,index,S+' columns');
end;

Var
  i,Count : Integer;
  m : TVideoMode;

begin
  InitVideo;
  InitKeyboard;
  Count:=GetVideoModeCount;
  For I:=1 to Count do
    begin
    GetVideoModeData(I-1,M);
    DumpMode(M,I-1);
    end;
  TextOut(1,Count+1,'Press any key to exit');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

35.5.10 GetVideoModeData

Synopsis: Get the specifications for a video mode

Declaration: `function GetVideoModeData(Index: Word;var Data: TVideoMode) : Boolean`

Visibility: default

Description: `GetVideoModeData` returns the characteristics of the `Index`-th video mode in `Data`. `Index` is zero based, and has a maximum value of `GetVideoModeCount-1`. If the current driver does not support setting of modes (`GetVideoModeCount=1`) and `Index` is zero, the current mode is returned.

The function returns `True` if the mode data was retrieved succesfully, `False` otherwise.

For an example, see GetVideoModeCount (1611).

Errors: In case `Index` has a wrong value, `False` is returned.

See also: GetVideoModeCount (1611), SetVideoMode (1615), GetVideoMode (1611)

35.5.11 InitVideo

Synopsis: Initialize video driver.

Declaration: `procedure InitVideo`

Visibility: default

Description: `InitVideo` Initializes the video subsystem. If the video system was already initialized, it does nothing. After the driver has been initialized, the `VideoBuf` and `OldVideoBuf` pointers are initialized, based on the `ScreenWidth` and `ScreenHeight` variables. When this is done, the screen is cleared.

For an example, see most other functions.

Errors: if the driver fails to initialize, the `ErrorCode` variable is set.

See also: DoneVideo (1607)

35.5.12 LockScreenUpdate

Synopsis: Prevent further screen updates.

Declaration: `procedure LockScreenUpdate`

Visibility: default

Description: `LockScreenUpdate` increments the screen update lock count with one. As long as the screen update lock count is not zero, UpdateScreen (1616) will not actually update the screen.

This function can be used to optimize screen updating: If a lot of writing on the screen needs to be done (by possibly unknown functions), calling `LockScreenUpdate` before the drawing, and UnlockScreenUpdate (1615) after the drawing, followed by a UpdateScreen (1616) call, all writing will be shown on screen at once.

For an example, see GetLockScreenCount (1609).

Errors: None.

See also: UpdateScreen (1616), UnlockScreenUpdate (1615), GetLockScreenCount (1609)

35.5.13 SetCursorPos

Synopsis: Set write cursor position.

Declaration: `procedure SetCursorPos(NewCursorX: Word;NewCursorY: Word)`

Visibility: default

Description: `SetCursorPos` positions the cursor on the given position: Column `NewCursorX` and row `NewCursorY`. The origin of the screen is the upper left corner, and has coordinates `(0,0)`.

The current position is stored in the `CursorX` and `CursorY` variables.

Errors: None.

See also: SetCursorType (1614)

Listing: ./videoex/ex2.pp

```
program example2;

uses video,keyboard;

Var
  P,PP,D : Integer;
  K: TKeyEvent;

  Procedure PutSquare (P : INteger; C : Char);

  begin
    VideoBuf^[P]:=Ord(C)+($07 shl 8);
    VideoBuf^[P+ScreenWidth]:=Ord(c)+($07 shl 8);
    VideoBuf^[P+1]:=Ord(c)+($07 shl 8);
    VideoBuf^[P+ScreenWidth+1]:=Ord(c)+($07 shl 8);
```

```
  end;

begin
  InitVideo;
  InitKeyBoard;
  P:=0;
  PP:=-1;
  Repeat
    If PP<>-1 then
      PutSquare(PP,' ');
    PutSquare(P,'#');
    SetCursorPos(P Mod ScreenWidth,P div ScreenWidth);
    UpdateScreen(False);
    PP:=P;
    Repeat
      D:=0;
      K:=TranslateKeyEvent(GetKeyEvent);
      Case GetKeyEventCode(K) of
        kbdLeft : If (P Mod ScreenWidth)<>0 then
                    D:=-1;
        kbdUp : If P>=ScreenWidth then
                  D:=-ScreenWidth;
        kbdRight : If ((P+2) Mod ScreenWidth)<>0 then
                     D:=1;
        kbdDown : if (P<(VideoBufSize div 2)-(ScreenWidth*2)) then
                    D:=ScreenWidth;
      end;
    Until (D<>0) or (GetKeyEventChar(K)='q');
    P:=P+D;
  until GetKeyEventChar(K)='q';
  DoneKeyBoard;
  DoneVideo;
end.
```

35.5.14 SetCursorType

Synopsis: Set cursor type

Declaration: `procedure SetCursorType(NewType: Word)`

Visibility: default

Description: `SetCursorType` sets the cursor to the type specified in `NewType`.

crHiddenHide cursor

crUnderLineUnderline cursor

crBlockBlock cursor

crHalfBlockHalf block cursor

Errors: None.

See also: SetCursorPos (1613)

35.5.15 SetVideoDriver

Synopsis: Install a new video driver.

Declaration: `function SetVideoDriver(const Driver: TVideoDriver) : Boolean`

Visibility: default

Description: `SetVideoDriver` sets the videodriver to be used to `Driver`. If the current videodriver is initialized (after a call to `InitVideo`) then it does nothing and returns `False`.

A new driver can only be installed if the previous driver was not yet activated (i.e. before a call to InitVideo (1612)) or after it was deactivated (i.e after a call to `DoneVideo`).

For more information about installing a videodriver, see viddriver (1597).

For an example, see the section on writing a custom video driver.

Errors: If the current driver is initialized, then `False` is returned.

See also: viddriver (1597)

35.5.16 SetVideoMode

Synopsis: Set current video mode.

Declaration: `function SetVideoMode(const Mode: TVideoMode) : Boolean`

Visibility: default

Description: `SetVideoMode` sets the video mode to the mode specified in `Mode`:

If the call was succesful, then the screen will have `Col` columns and `Row` rows, and will be displaying in color if `Color` is `True`.

The function returns `True` if the mode was set succesfully, `False` otherwise.

Note that the video mode may not always be set. E.g. a console on Linux or a telnet session cannot always set the mode. It is important to check the error value returned by this function if it was not succesful.

The mode can be set when the video driver has not yet been initialized (i.e. before InitVideo (1612) was called) In that case, the video mode will be stored, and after the driver was initialized, an attempt will be made to set the requested mode. Changing the video driver before the call to `InitVideo` will clear the stored video mode.

To know which modes are valid, use the GetVideoModeCount (1611) and GetVideoModeData (1612) functions. To retrieve the current video mode, use the GetVideoMode (1611) procedure.

Errors: If the specified mode cannot be set, then `errVioNoSuchMode` may be set in `ErrorCode`

See also: GetVideoModeCount (1611), GetVideoModeData (1612), GetVideoMode (1611)

35.5.17 UnlockScreenUpdate

Synopsis: Unlock screen update.

Declaration: `procedure UnlockScreenUpdate`

Visibility: default

Description: `UnlockScreenUpdate` decrements the screen update lock count with one if it is larger than zero. When the lock count reaches zero, the UpdateScreen (1616) will actually update the screen. No screen update will be performed as long as the screen update lock count is nonzero. This mechanism can be used to increase screen performance in case a lot of writing is done.

It is important to make sure that each call to LockScreenUpdate (1613) is matched by exactly one call to `UnlockScreenUpdate`

For an example, see GetLockScreenCount (1609).

Errors: None.

See also: LockScreenUpdate (1613), GetLockScreenCount (1609), UpdateScreen (1616)

35.5.18 UpdateScreen

Synopsis: Update physical screen with internal screen image.

Declaration: `procedure UpdateScreen(Force: Boolean)`

Visibility: default

Description: `UpdateScreen` synchronizes the actual screen with the contents of the `VideoBuf` internal buffer. The parameter `Force` specifies whether the whole screen has to be redrawn (`Force=True`) or only parts that have changed since the last update of the screen.

The `Video` unit keeps an internal copy of the screen as it last wrote it to the screen (in the `OldVideoBuf` array). The current contents of `VideoBuf` are examined to see what locations on the screen need to be updated. On slow terminals (e.g. a linux telnet session) this mechanism can speed up the screen redraw considerably.

On platforms where mouse cursor visibility is not guaranteed to be preserved during screen updates this routine has to restore the mouse cursor after the update (usually by calling HideMouse from unit Mouse before the real update and ShowMouse afterwards).

For an example, see most other functions.

Errors: None.

See also: ClearScreen (1606)

Chapter 36

Reference for unit 'wincrt'

36.1 Overview

The wincrt unit provides some auxiliary routines for use with the graph (570) unit, namely keyboard support. It has no connection with the crt (379) unit, nor with the Turbo-Pascal for Windows WinCrt unit. As such, it should not be used by end users. Refer to the crt (379) unit instead.

36.2 Constants, types and variables

36.2.1 Variables

```
directvideo : Boolean
```

On windows, this variable is ignored.

```
lastmode : Word
```

Is supposed to contain the last used video mode, but is actually unused.

36.3 Procedures and functions

36.3.1 delay

Synopsis: Pause program execution

Declaration: `procedure delay(ms: Word)`

Visibility: default

Description: `Delay` stops program execution for the indicated number `ms` of milliseconds.

See also: sound (1618), nosound (1618)

36.3.2 keypressed

Synopsis: Check if a key was pressed.

Declaration: `function keypressed : Boolean`

Visibility: default

Description: `KeyPressed` returns `True` if the user pressed a key, or `False` if not. It does not wait for the user to press a key.

See also: readkey (1618)

36.3.3 nosound

Synopsis: Stop the speaker

Declaration: `procedure nosound`

Visibility: default

Description: `NoSound` does nothing, windows does not support this.

See also: sound (1618)

36.3.4 readkey

Synopsis: Read a key from the keyboard

Declaration: `function readkey : Char`

Visibility: default

Description: `ReadKey` reads a key from the keyboard, and returns the ASCII value of the key, or the scancode of the key in case it is a special key.

The function waits until a key is pressed.

See also: KeyPressed (1617)

36.3.5 sound

Synopsis: Sound PC speaker

Declaration: `procedure sound(hz: Word)`

Visibility: default

Description: `Sound` sounds the PC speaker. It emits a tone with frequency `Hz` for 500 milliseconds. (the time argument is required by the windows API)

See also: nosound (1618)

36.3.6 textmode

Synopsis: Set indicated text mode

Declaration: `procedure textmode(mode: Integer)`

Visibility: default

Description: `TextMode` does nothing.

Chapter 37

Reference for unit 'x86'

37.1 Used units

Table 37.1: Used units by unit 'x86'

Name	Page
BaseUnix	94

37.2 Overview

The x86 unit contains some of the routines that were present in the 1.0.X Linux unit, and which were Intel (PC) architecture specific.

These calls have been preserved for compatibility, but should be considered deprecated: they are not portable and may not even work on future linux versions.

37.3 Procedures and functions

37.3.1 fpIOperm

Synopsis: Set permission on IO ports

Declaration: `function fpIOperm(From: Cardinal;Num: Cardinal;Value: cint) : cint`

Visibility: default

Description: `FpIOperm` sets permissions on `Num` ports starting with port `From` to `Value`. The function returns zero if the call was successfull, a nonzero value otherwise.

Note:

- This works ONLY as root.
- Only the first `0x03ff` ports can be set.
- When doing a FpFork (142), the permissions are reset. When doing a FpExecVE (139) they are kept.

Errors: Extended error information can be retrieved with FpGetErrno (145)

37.3.2 fpIoPL

Synopsis: Set I/O privilege level

Declaration:
```
function fpIoPL(Level: cint) : cint
```

Visibility: default

Description: `FpIoPL` sets the I/O privilige level. It is intended for completeness only, one should normally not use it.

37.3.3 ReadPort

Synopsis: Read data from a PC port

Declaration:
```
procedure ReadPort(Port: LongInt;var Value: Byte)
procedure ReadPort(Port: LongInt;var Value: LongInt)
procedure ReadPort(Port: LongInt;var Value: Word)
```

Visibility: default

Description: `ReadPort` reads one Byte, Word or Longint from port `Port` into `Value`.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), ReadPortB (1620), ReadPortW (1621), ReadPortL (1621), WritePort (1621), WritePortB (1622), WritePortL (1622), WritePortW (1622)

37.3.4 ReadPortB

Synopsis: Read bytes from a PC port

Declaration:
```
function ReadPortB(Port: LongInt) : Byte
procedure ReadPortB(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortB` reads `Count` bytes from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` bytes.

The functional form of `ReadPortB` reads 1 byte from port `B` and returns the byte that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), ReadPort (1620), ReadPortW (1621), ReadPortL (1621), WritePort (1621), WritePortB (1622), WritePortL (1622), WritePortW (1622)

37.3.5 ReadPortL

Synopsis: Read longints from a PC port

Declaration:
```
function ReadPortL(Port: LongInt) : LongInt
procedure ReadPortL(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortL` reads `Count` longints from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` Longints.

The functional form of `ReadPortL` reads 1 longint from port `B` and returns the longint that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), ReadPort (1620), ReadPortW (1621), ReadPortB (1620), WritePort (1621), WritePortB (1622), WritePortL (1622), WritePortW (1622)

37.3.6 ReadPortW

Synopsis: Read Words from a PC port

Declaration:
```
function ReadPortW(Port: LongInt) : Word
procedure ReadPortW(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortW` reads `Count` words from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` words.

The functional form of `ReadPortW` reads 1 word from port `B` and returns the word that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), ReadPort (1620), ReadPortB (1620), ReadPortL (1621), WritePort (1621), WritePortB (1622), WritePortL (1622), WritePortW (1622)

37.3.7 WritePort

Synopsis: Write data to PC port

Declaration:
```
procedure WritePort(Port: LongInt;Value: Byte)
procedure WritePort(Port: LongInt;Value: LongInt)
procedure WritePort(Port: LongInt;Value: Word)
```

Visibility: default

Description: `WritePort` writes `Value` – 1 byte, Word or longint – to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), WritePortB (1622), WritePortL (1622), WritePortW (1622), ReadPortB (1620), ReadPortL (1621), ReadPortW (1621)

37.3.8 WritePortB

Synopsis: Write byte to PC port

Declaration:
```
procedure WritePortB(Port: LongInt;Value: Byte)
procedure WritePortB(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), WritePort (1621), WritePortL (1622), WritePortW (1622), ReadPortB (1620), ReadPortL (1621), ReadPortW (1621)

37.3.9 WritePortl

Synopsis: Write longint to PC port.

Declaration:
```
procedure WritePortL(Port: LongInt;Value: LongInt)
procedure WritePortl(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), WritePort (1621), WritePortB (1622), WritePortW (1622), ReadPortB (1620), ReadPortL (1621), ReadPortW (1621)

37.3.10 WritePortW

Synopsis: Write Word to PC port

Declaration:
```
procedure WritePortW(Port: LongInt;Value: Word)
procedure WritePortW(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (1619) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (1619), WritePort (1621), WritePortL (1622), WritePortB (1622), ReadPortB (1620), ReadPortL (1621), ReadPortW (1621)