XTDFPC supplied units : Reference guide.

Reference guide for additional units package XTDFPC .

1.0

March 29, 1999

Marco van de Voort
Thomas Schatzl

Contents

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the `XTDFPC` units

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

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

The cross-references come in two flavors:

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

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

Credits

A genuine thanks goes to the following people:

- Michael Van Cannëyt for the original FPC docs styles and layout, which I cowardly use.
- Peter Vreman for some help with the documentation and helping out with other XTDFPCrelated questions.
- Thomas Schatzl for allowing to distribute his units in XTDFPC

Chapter 1

XtdFPC package/ The Toolkit

This chapter contains all general information about XTDFPC, the other chapters are dedicated to a specific unit.

1.1 Installation

1.1.1 Requirements

- All requirements of FPC for the platform/OS you use.
- A recent version. It's still impossible to keep an archive with lowlevel stuff like this up to date with all versions. Preferably a recent devellopers snapshot, but mosttimes the last release also works, or only with minor adjustments.
- To run the assembler: An i386 system. Preferably Linux, Go32V2 or Win32.

1.1.2 platforms

The platforms supported are (with a recent snapshot)

- Go32V2, the main target.
- Linux, the other target, but generally the compiler has more problems on this OS, so sometimes the newest quirks haven't been solved on Linux.
- Win32, experimental, because Win32 doesn't fully support all Turbo units. Most units will work, some not, or will require minor adjustments.
- Win32/Delphi, I want to try to get SOME units Delphi compatible somewhere in the future.
- others untested.
- Borland Pascal used to be a target, but I abandoned it. A lot of the routines should still work with it.

1.1.3 FPCversions

It's hard to keep a source-archive compatible with every version and target, because the FPC compiler (and RTL) still change slowly but steadily.

As said before, I try to keep the units compatible with the last release and the last snapshot. If that isn't possible, I choose for compability with the last snapshot, and mainly the Linux and Go32V2 targets.

I don't say that it won't run on the other versions and targets, or that I don't support those other targets. However, I can't test on those targets myself, or it takes up too much time and everything for those targets is heresay, and untested.

The Borland Pascal 7.0 compability has been dropped, but single units might still be Borland compilable with a reasonable effort. Turn assembler off! (syseq should do that automagically).

Also often you have to turn of assembler for older snapshots, because the assembler reader is one of the things that changes most (at least until 0.99.10).

1.2 Internals

1.2.1 Conditionals

i386 Intelstyle processor(also AMD, Cyrix and NextGen), required for using the assembler. If you don't $DEFINE this, the libraries will revert to the pascal-alternatives

Linux This conditional is normally defined by the compiler, so there is no need to define it yourself except when crosscompiling.

Turbo Syseq.inc tests VER60 and VER70 defines (which are automatically set by BP/TP), and if one such define is found, syseq.inc $DEFINEs Turbo, so you'll know that this is Borland or Turbo Pascal specific code. Mainly used to avoid importing Go32 under TP, and to turn off assembler (Since it's 32bit it won't work under TP/BP)

CloseFind Indicates if Dos.FindClose is used after each Dos.FindFirst. Will be always true in the future, except under Turbopascal. It's true under Go32V2 because the Win9x LFN system also requires FindFirsts to be closed) (You can undefine it for DOS, if you have compiled your RTL without LFN support, the so called RTL-Lite), or if you only use plain DOS.

UseAsm This define is local to the unit it's in. If you undefine it, no assembler will be used in that unit, even if i386 is defined.

OldLinuxWin Used to select between two type of Crts. TRUE for version 0.99.8 and older, false for 0.99.9 and newer The new Linux Crt has a pointer to the virtual screen and the screen-dimensions in the definition, for older versions you have to alter the Crt before EWindow works under Linux. If your 0.99.9 doesn't work, you'll have to upgrade to a newer snapshot or edit syseq.inc

1.2.2 Procedure and unit names

The names of the procedures aren't fixed yet. For now, I have stuck to the names in the Modula2-version of XtdFPC. If somebody has a coherent namesystem, with solid arguments in favour of the new system, I'm principally willing to change the procedure names. Often, procedurenames are based on SWAG, and already a bit standard.

This will have to happen fast. At the time of this (0.10) release, XtdFPC is downloaded almost daily, and I don't want to change all procedure names after a lot of people based their programs on the current names.

1.3 Copyright and license data

This is the file COPYING.TXT blended into the documentation, it applies to the XTDLIB/FPC toolkit and its documentation: source files copyrighted by Marco van de Voort.

The source code of XtdFPC is distributed under the GNU General Public License (see next subsection) with the following exceptions:

Object files and libraries linked into an application may be distributed without source code, as long as the application is not commercial. Commercial use requires a fee. Mail Marcov@stack.nl for more details.

If you didn't receive a copy of the file COPYING, contact:

```
Free Software Foundation
675 Mass Ave
Cambridge, MA  02139
USA
```

Suggestions, ideas ?? Please correct spelling mistakes in the license, if you see one.

1.3.1 The GNU General Public Licence

The following is the text of the GNU General Public Licence, under the terms of which this software is distrubuted.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

1.4 These docs

These docs are written in , with a slightly enhanced fpc.sty (which is the FPC-compilers style file).

Currently the quality is about the same as the handcoded HTML docs. The lay out is a bit better, however the handmade docs have been testread and corrected. The converted own documentation looks better than Tom’s, but that is because I did Tom’s docs with a script, and mine by hand. My documentation was also created to mimic the Free Pascal scheme, so it was easier to convert (the huge bunch of underscores in Tom’s DPMI unit was pure horror)

Also I’m still playing with the tables of the dpmi unit.

The main reason for documentation is however the better structure, which makes it easier to enhance the documents, and even a crossreference with the units of the FPC Unitsreference is possible.

Some of the advantages:

- Because almost all procedure, function, constant, type and variable reference sections are implemented as so called environment-macros, a small update to the definition of such base environment causes all referencesections build on such environment to be also updated.

  So if I wish to say put the declaration part of each procedure subsection in a different, smaller font, I simply add

  ```
  \small
  ```

  to the declaration of the procedureenvironment, and all procedure sections will change.

- Currently, the documentation is released as PDF. However the (distributes) sources compile and generate a nice DVI (or PS) if you prefer that. I tried to generate HTML, but it is all too sensitive. Maybe at some later point when I now better.

Chapter 2

ECRC32 unit

2.1 general notes

ECRC32, fast, easy to use CRC-32 checksum routines

A very small and simple unit which provides fast CRC-32 routines, probably faster than most 16-bit routines, because using 32-bits assembler/variables eases programming a unit like this considerably.

At the moment of writing, the Pascal version of CalcCrc wasn't operational, because of a problem in the open-array handling. Please check the header of the unit for more information.

This unit has only one example demonstrating most procedures, InvertCRC (21)

2.2 Types and Constants

2.2.1 Poly32

Declaration: `CONST Poly32 :  CARDINAL = $0EDB88320;`

Description: Poly32 is a 32bits seedvalue for the CreateCRC32Table (20) procedure.

This is the standard polynome, used by 99.9% of the programmers. The CRC's you see when you list archives (pkunzip v or arj l), or in WinZip, are all created with this polynomal. If you stick to the procedure used in the example for this unit, you will get the same crcvalue as e.g. ARJ for a certain file.

The only reason not to use this standard polynomal is for antihacking purposes.

See also: CreateCRC32Table (20)

2.3 procedures and functions

2.3.1 CreateCRC32Table

Declaration: `PROCEDURE CreateCRC32Table( Poly32 :  CARDINAL);`

Description: This procedure `must` be called prior to any use of the CRCcalculating routines.

The procedure generates an internal, 256 elements big table of 32bits CRCcodes on the heap which is needed for fast byte/characterbased CRC calculations. The parameter is a certain 32bits seed (corresponding with a polynomal I believe) which is used to create the table.

If you want your CRC's to be compatible with other programs (like ARJ,ZIP and dozens of others), you should use Poly32 (20) as parameter

The generation of the table is very fast, and only has to be done once. (but more calls to CreateCRC32Table are allowed).

See also: Poly32 (20)

2.3.2 CalcCRC

Declaration: `FUNCTION CalcCRC (crc:  CARDINAL;Buffer;BufSize:CARDINAL) : CARDINAL;`

Description: Calculates a CRC32 checksum value over the first BufSize bytes in Buffer.

The function returns the calculated CRC value. The crc parameter, is equal to $FFFFFFF if you use the procedure for the first time, or equal to the value returned by a previous call to CalcCRC. See the example for more details.

For performance reasons CalcCRC doesn't call AddCRC, the AddCRC code is included in CalcCRC

See also: AddCRC (21) InvertCRC (21)

2.3.3 AddCRC

Declaration: `PROCEDURE AddCRC( VAR crc :  CARDINAL;ch :  BYTE );`

Description: Calculates a CRC32 over one byte, updates value CRC (from previous AddCRC or CalcCRC calls, or $0FFFFFFFF if this is the first byte) to a new CRC value Not really meant to be called in a loop, use CalcCRC (21) to calculate a CRC over a big memory chunk.

Some structures add 4 zerobytes to the end of the to be crc'ed data. AddCrc is meant to calculate a CRC over those 4 zero bytes if you can't simply store 4 extra zero's after the data

See also: CalcCRC (21)

2.3.4 InvertCRC

Declaration: `FUNCTION InvertCRC(crc :  CARDINAL ):CARDINAL;`

Description: XOR's a crcvalue with 1. If you calculate a crc using CalcCRC, you should use this procedure on the value to obtain a standard CRC.

See also: none.

2.3.5 example

This is an example how to calculate a CRC over a file on disk. You can pack the same file with ARJ or ZIP, and use Winzip, pkunzip -v or ARJ l to view the CRC-value calculated by the compressor, and compare it to the one generated by this example.

```
PROGRAM CRCTEST;              { Tested }

USES ECrc32,EFIO;            { EFIO is only used to display a value in HEX form}

CONST FileName='test.dat';        {Make sure the file exists, and size>0}

VAR F           : File;
    Buffer      : ARRAY[0..2047] OF BYTE; { 2kb buffer, but CalcCRC
                                            can process a Gigabyte buffer}
    BytesRead   : WORD;
    Crc         : CARDINAL;

BEGIN
 CreateCRC32Table(Poly32);                { Generate the CRC table}
 Assign(F,Filename);
 Reset(F,1);
 CRC:=$0FFFFFFFF;                         { Standard start-value}
 REPEAT
  BlockRead(F,Buffer,2048,BytesRead);     { Read a chunk}
  Crc:=CalcCrc(CRC,Buffer,BytesRead);     { calculate a CRC over the chunk}
 UNTIL BytesRead<>2048;                   { Until end-of-file }
 Close(F);
 WrLngHex(InvertCRC(CRC));                { Invert CRC and display}
 Writeln;
END.
```

Chapter 3

EDate Unit

EDate, fast and advanced date handling

EDate is a Date unit in plain 32bit AT&T assembler, and should be fast enough for most applications. The originals were used to process dates in logcompression and loganalysers, which is the reason the procedure were written in assembler (386SX-20!). Of course there are Pascal equivalents.

All date procedures have been checked for problems around 2000. Some old routines didn't consider the year 2000 to be a leapyear. Fixed.

3.1 Types and constants

3.1.1 Months and Dows

Declaration:
```
TYPE     MonthStr    = ARRAY[1..12] OF String[3];
          DowsStr     = ARRAY[0..6] OF String[3];

CONST
   Months:MonthStr=('Jan','Feb','Mar','Apr','May','Jun','Jul',
                    'Aug','Sep','Oct','Nov','Dec');
```

Description: The constants used are the days of the week, and the abbreviations of the months. You can change these if you wish, but keep them exactly 3 characters long, since these constants are used by DatiToStr (25). (Or fix DatiToStr too)

SeeAlso DatiToStr (25)

3.1.2 Ceremony

Declaration:
```
TYPE     Ceremony =    (AshWednesday,GoodFriday,EasterSunday,
                        EasterMonday,AscensionDay,Whitsunday,
                        WhitsunMonday,CorpusChristiDay,RepentanceDay,
                        FirstAdvent,SecondAdvent,MothersDay);
```

Description: This type is used by MovableCeremony (26), and contains all the holidays that MovableCeremony can return a date for.

SeeAlso MovableCeremony (26)

3.2 Procedures and functions

3.2.1 LeapYr

Declaration: `FUNCTION LeapYr( Year : WORD ) : BOOLEAN;`

Description: Returns TRUE when the `Year` is a leapyear. FALSE otherwise.
A leapyear is defined as (Year and 3=0) AND ((Year DIV 100)<>0 OR (Year DIV 400)=0) which will be correct until the year 4000 or so.

See also: Used by most other procedures.

3.2.2 DayNr

Declaration: `FUNCTION DayNr( Day,Month,Year : WORD) : WORD;`

Description: Calculates daynumber, January 1st =1. Februari 1st=32, tests for leapyears. (With LeapYr (24)) Easy way to subtract days within a year, or (DayNr(31,12,Year-1) is the number of days in last year. Use DayNrBack (24) to convert a day number back to a real date

Note: Remember, a DayNr is NOT corresponding to a date. A year and a DayNr does though.

See also: LeapYr (24)

3.2.3 DayNrBack

Declaration: `PROCEDURE DayNrBack( Year,DayNr:WORD; VAR Month,Day : WORD);`

Description: Converts a daynumber(e.g. created with DayNr (24)) back to day and month format.

Note: Expires Feb 28th, 2100

See also: DayNr (24)

3.2.4 DOW

Declaration: `FUNCTION DOW( Year,Month,Day:WORD): WORD;`

Description: Day of week (Sunday,Monday etc) for a given date. DOW(a Sunday)=0; DOW(a Monday)=1 etc.

Note: Expires Feb 28th, 2100

See also: WeekNr (25), Easter (25)

3.2.5 ToUnix

Declaration: `FUNCTION ToUnix(Year, Month, Day, Hrs, Mins, Secs : WORD):CARDINAL;`

Description: Calculates unixdate, which is defined as the number of seconds after 1-1-70.

Notes:
Expires Feb 28th, 2100
This value is big endian, while some platforms use little endian unix dates. So if you want to compare your calculated unix date with a date retrieved from a filesystem, make sure that the dates are both in the same 'endian' format.

See also: DayNr (24) and (via DayNr) LeapYr (24) FromUnix (25)

3.2.6 FromUnix

Declaration: `PROCEDURE FromUnix(Unix : LONGINT;VAR Year,Month,Day,Hour,Min,Sec:WORD);`

Description: Retrieves DDMMYY HH:MM:SS back the from unixdate, which is defined as the number of seconds after 1170.

Notes:
Expires Feb 28th, 2100
All values are big endian, while some platforms use little endian unix dates. So if you want to compare your calculated unix date with a date retrieved from a filesystem, make sure that the dates are both in the same 'endian' format.

See also: ToUnix (24) LeapYr (24)

3.2.7 WeekNr

Declaration: `FUNCTION WeekNr(Year,Mnth,Day:WORD):WORD;`

Description: Returns weeknumber (in year) for a certain date. Remember this isn't simply DayNr (24) DIV 7. Jan 1st is not necessarily the first day of week 1.

Note: Expires Feb 28th, 2100, because DOW does.

See also: DOW (24) DayNr (24)

3.2.8 Easter

Declaration: `FUNCTION Easter (Year : WORD) : WORD;`

Description: Returns the date of Easter for a given year. 1=March 1st. 32= April 1st etc.

Note: Expires Feb 28th, 2100, because DOW does.

See also: DOW (24)

3.2.9 DatiToStr

Declaration:
```
PROCEDURE DatiToStr(Hour,Min,Sec,Day,Month,Year:WORD; CONST Format:String;
VAR DateStr : String);
```

Description: DatiToStr is a complex configurable date-formatting routine, and the biggest EDate procedure by far.

Basically it copies `Format` to `DateStr`, while replacing some switches. Padding character is default (when the procedure starts) 0, but can be changed to space

Format-string to see all options, use this string twice (once prefixed with %i to see padding) to see all options: '%H:%M:%S %h:%m:%s %U:%m %a %u:%m %A %D-%O-%Y %d-%o-%y, %W %J %D%L, %y';

Note: Make sure `DateStr` is big enough
If you don't have day switches in your Format-string, you can specify anything for day.

See also: DOW (24)

Table 3.1: DatiToStr options

Replaces	with	padding with padchar?
%H	Hour	no
%M	Min	no
%S	Sec	no
%D	Day	no
%O	Month	no
%Y	Year MOD 100	Yes
%h	Hour	Yes
%m	Min	Yes
%s	Sec	Yes
%d	Day	Yes
%o	Month	Yes
%Y	Day	Yes
%y	Year (all 4 characters)	no
%J or %j	3 character month (Jan,Feb)	n/a
%W or %w	3 character day of week (Sun,Mon)	n/a
%L	suffix of day (st,nd,rd,th)	n/a
%i	padding character to space	No output,internal
%I	padding character to zero	No output,internal
%U	Hour MOD 13 (1..12)	No
%u	Hour MOD 13 (1..12)	Yes
%A	AM or PM	n/a
%a	am or pm	n/a

3.2.10 MovableCeremony

Declaration: `PROCEDURE MovableCeremony (MovableCeremon : Ceremony (23); Year: WORD; VAR Day, Month : WORD);`

Description: Returns `day` and `month` of a certain movable ceremony in a certain `Year`

Note: Expirency date not known, but based on DayNr and LeapYr, so it can probably be used in the 21st century.

See also: DOW (24), DayNr (24), DayNrBack (24)

```
Uses EDate;

VAR Day, Month : LONGINT;

BEGIN
  MovableCeremony(AscensionDay, 1998, Day, Month);
  Writeln('AscensionDay : ', Day, '-', Month, '-1998');
END.
```

Chapter 4

The EDirTree unit.

4.1 Introduction

This is the documentation for `EDirtree` unit which is one of the three units for recursive directory scanning right now:

- This unit, `EDirTree` is the oldest one, and also the most used one.
- `ODirTree` implements the same functionality as `EDirTree`, and even a bit more in an OOP modelled way.
- chapter **??** is basically the same unit as EDirTree, only it uses Linux globbing and FStat to get its data. Also all structures use the stat record instead of SearchRec

This is both the manual for EDirTree and EDirGlob, which are on an interface level nearly the same, except that in all EDirGlob's procedure and types SearchRec is replaced by Linux.stat.

The main target is FPC (Linux and Go32V2 tested). How I search directories on the Win32 platform I don't know. I would like to implement that in a more or less Delphi compatible fashion. Anyone?

4.1.1 Objectives while implementing the unit

When I was developing this unit, I kept some things in mind:¡p¿

- One basic unit for all scanning for files and directories, meant for when one FindFirst/FindNext loop wouldn't do the job anymore.
- A simple interface which eased extending existing simple programs with directory recursion.
- Using a the same unit over and over again, no template unit which you have to adapt to your fresh application everytime you use it. In general, this meant using procedure-variables or OOP, to allow application depend code to be used internally in the unit.

- The tree-building in memory routines that can be used for everything. I didn't implement this yet, but it bears down to an untyped pointer for every file/directory so you can add your own information to the tree. I did implement this in ODirTree.

The things I wanted to do with this unit are also simple:

- I wanted to make a kind of automatic indexer for my home-made archive-CDs. The program should read 4DOS-DESCRIPT.ION's, and generated files.bbses and a main-index. I wanted to implement a 00-index.txt and 00index.html system on top of that, but the longfilesupport became more important, and I never extended the program.
- The other reason was to have a custom scanning module. I use a modified version of FileFind to scan for ARJ and other archive types on the harddisk. This procedure however looks IN the files for detection, the extension doesn't matter (findarch.pp)

EDirTree now uses forward slashes (if Linux exists) and FindClose (almost always necessary).

4.1.2 procedure groups

EDirTree has two separate groups of procedures, each corresponding with a different kind of directory scan:

- FileScan (31) is the most used procedure. FileScan implements recursive scanning of a certain directory and below, and a procedure supplied by is run for each file which matches a certain pattern (like *.pp). Depending on the boolean seevarDirsToo matching directorynames will also be reported to this procedure.
- The second set of procedures creates and operates on a directory tree read into memory
  - BuildTree (32) creates the binary tree and scans the specified path and adds all its directories (recursive) to a branching tree in memory.
  - SearchForFiles (32) scans the path again and adds all matching files to the tree. SearchForFiles can be run multiple times for different patterns
  - ScanTree (32) scans the tree (in memory) and executes a procedure (one of yours) for each file which matches the wildcard. ScanTree can also run a procedure on each directory if DirsToo (30) is TRUE
  - Finally KillFileTree (33) removes a tree built by the previous procedures from memory.

Two other procedures exist and are used by both sets procedures.

- SetFAttr (30) Sets fileattributes used by the Dos.FindFirst and Dos.FindNext procedures in the above procedures. The directory bit is not important
- ClearStat (31) is the initialisation code of the unit. It resets all important parameters to their default values (DirsToo (30), file attributes, and all the statistics in the variables section

4.2 Types and constants

4.2.1 procedure types

Declaration:
```
ReportProc = PROCEDURE (Path:PathStr;Search:SearchRec);
FileProc = PROCEDURE (P:PathStr;Search:SearchRec);
DirProc = PROCEDURE (P:PathStr);
DetectProc = FUNCTION (Search:SearchRec;P:PathStr):BOOLEAN;
```

Description: These proceduretypes accept one or two parameters.

- The PathStr typed parameter is always the path of the file the procedure has to process WITHOUT the filename itself.
- The SearchRec parameter (the record used by FindFirst,FindNext, see Dos unit for declaration) describes the found file (Search.Name is it's name, Search.Size it's size in bytes and so on).

The DirProc has only one parameter since it operates on a directory. The DetectProc type is a function, and the return value indicates if the file should be added to the tree (TRUE) or not (FALSE).

See also: FileScan (31), BuildTree (32), SearchForFiles (32), ScanTree (32)

4.2.2 FilAttr

Declaration:
```
FilAttr = BYTE;
```

Description: This type is meant to type the attributes for FindFirst and FindNext. A remnant of the units Modula-2 origin where it is a SET type.

See also: SetFAttr (30)

4.2.3 TreeBuildingTypes

Declaration:
```
Fileptr = ^FilesRec; FilesRec = RECORD Next :  FilePtr; next file in this
directory DirE : SearchRec; Unit DOS, record for findfirst END;

DirTreePoint = ^DirTreeRecord; DirTreeRecord = RECORD NextDir, next directory
on this level SubDirs :  DirTreePoint; subdirectories (Lower than this
level) Name :  PathStr; Name of directory Files :  FilePtr; see above
END;
```

Description: These structures are used to build a tree (See BuildTree (32)) in memory.

(Here is an image missing)

Horizontally, all directories are on the same level, horizontal lines indicate the NextDir pointer of DirTreeRecord, vertical lines equal the SubDirs pointer of DirTreeRecord. Directory DOS has no subdirectories, directory Windows two (System AND INF). Directory C: is the top level, and has two directories. All non-used pointers are nil.

Files aren't included in this picture. Imagine every directory having a single linked list of files in the direction perpendicular to the screen.

See also: BuildTree (32), SearchForFiles (32), ScanTree (32)

4.3 Variables

4.3.1 DirsToo

Declaration: `DirsToo : BOOLEAN`

Description: If this boolean variable is true, then also directorynames which match the search-pattern of FileScan (31) or ScanTree (32) are reported to the Report procedure-variable.

See also: FileScan (31) or ScanTree (32)

4.3.2 Statisticsvariables

Declaration: `FoundCount : LONGINT; Number of files found TotalBytes : LONGINT; Total bytes in files found. See also` ClusterSize (30) `FoundDirs : LONGINT; Directories found ( . and .. are ignored)`

Description: These variables contain some statistics about the last search, or last series of searches. The statistics are cleared by ClearStat (31)

See also: ClearStat (31), ScanTree (32), FileScan (31)

4.3.3 ClusterSize

Declaration: `ClusterSize : LONGINT`

Description: This variable can be used to control rounding of filesizes before the Statisticsvariables (30) are updated. If ClusterSize is zero, no rounding is done.

The idea is to be able to find a better estimate for the size of a selection files than simply adding up the filesizes, by taking clustersize (inode blocksize) into account. This estimate is better but not perfect (directory entries also take up space), but this is normally relatively small. Also it's quite simple to implement, since all filesystems have such a value.

Under Linux it's even more difficult, since directories can be complete mounted filesystems with a different inode size.

4.4 Procedures and Functions

4.4.1 SetFAttr

Declaration: `PROCEDURE SetFAttr (Attr:FilAttr);`

Description: Sets attributes used for all FindFirstFindNext couples in EDirTree. Directory attribute is added or cleared when necessary(if the program searches for directories or just for files).

Errors: None.

See also: FilAttr (29), FileScan (31), BuildTree (32), SearchForFiles (32)

```
Uses EDirTree;

BEGIN
 ClearStat;                      // Resets unit.
 SetFAttr(archive+readonly);     // include archive and readonly files in next search
END.
```

4.4.2 ClearStat

Declaration: `PROCEDURE ClearStat;`

Description: This is the initialiation code of the unit. It resets all the Statisticsvariables (30) to zero, DirsToo (30) to false and calls SetFAttr (30) to let the unit include all files except volume-IDs.

The initcode is moved to a procedure so mainprograms can reset the unit, and because TopSpeed modula-2 doesn't allow overlayed units to have initcode.

Errors: None.

See also: Statisticsvariables (30), DirsToo (30) and SetFAttr (30)

See SetFAttr (30) for an example.

4.4.3 FileScan

Declaration: `PROCEDURE FileScan(RootDir,FileName : PChar;Report: ReportProc (29));`

Description: Filesearch in path `RootDir` and in its subdirectories, for files matching `FileName` (may be a wildcard, directories are regarded as files when DirsToo (30)=TRUE). Files are reported to the procedure `Report`, with all information (path and Dos.SearchRec).

To quickly execute a procedure in every directory, enter "." as filename, and assign TRUE to DirsToo (30)

FileScan is quite powerfull, however if you want to do a very complex scan, or scan a certain drive or directory several (more than 2) times, look at the treebuilding procedures (ScanTree (32) SearchForFiles (32) and BuildTree (32))

Errors: None.

See also: ScanTree (32) SearchForFiles (32) , BuildTree (32) , ReportProc (29)

```
Uses EDirTree

PROCEDURE WriteOutput(Path:PathStr;FileData:SearchRec);

BEGIN
 Write(Path,FileData.name,'  ',FileData.size);
END;

BEGIN
 DirsToo:=FALSE;                          {EDirtree procedure, don't report
                                            directories, incase a directory with
                                            extension .pas exists}
 FileScan('c:\','*.pas',@WriteOutput);    {searches for *.pas on entire 'c:\'}
END.
```

4.4.4 BuildTree

Declaration: `FUNCTION BuildTree(CONST RootDir: PChar):` DirTreePoint (29);

Description: Searches path `RootDir` and adds all directories to a DirTreePoint (29) typed tree. A pointer to the created tree is returned.

Errors: None.

See also: ScanTree (32) SearchForFiles (32) , TreeBuildingTypes (29) KillFileTree (33)

The buildtree procedures don't have an example in the helpfile. See DirTest.pp in the XTDFPC package.

4.4.5 SearchForFiles

Declaration: `PROCEDURE SearchForFiles(Root:` DirTreePoint (29);`CONST Pattern:PChar;Select:` DetectProc (29));

Description: This procedure is used after BuildTree (32) (which creates the `Root` DirTreePointer (29)), and searches in all directories (found by BuildTree (32)) for an occurance of `Pattern`, and adds those files to the tree under the "files" field of all DirTreePoints (29).
`Pattern` is something like "*.txt"

`Select` is a function which you can supply to do additional checks. If this function returns TRUE the file will be added to the tree, if it returns FALSE it won't. If you don't want to use this feature, pass a procedure which always returns TRUE.

Can be used several times, for more than one extension/pattern, however overlapping patterns will result in duplicate files. (*.pp and helloworld.* will cause helloworld.pp to be added twice)

Errors: None.

See also: ScanTree (32) BuildTree (32) , TreeBuildingTypes (29) procedure types (29) KillFileTree (33)

The buildtree set of procedures don't have an example in the helpfile. See DirTest.pp in the XTDFPC package.

4.4.6 ScanTree

Declaration: `PROCEDURE ScanTree(Root :` DirTreePoint (29);`DoFile:` FileProc (29); `DoDir:` DirProc (29));

Description: Use this procedure after a BuildTree (32) and optionally one or more SearchForFiles (32).

This procedure scans the directory tree `Root` and runs `DoFile` for each found file in the tree. `DoDir` is also run for every directory when DirsToo (30)=TRUE.

Errors: None.

See also: BuildTree (32), TreeBuildingTypes (29), procedure types (29), DirsToo (30), KillFileTree (33), SearchForFiles (32)

The buildtree set of procedures don't have an example in the helpfile. See DirTest.pp in the XTDFPC package.

4.4.7 KillFileTree

Declaration: `PROCEDURE KillFileTree(VAR Root: DirTreePoint (29));`

Description: Use after a BuildTree (32) and (optionally) one or more SearchForFiles (32) calls. This procedure simply removes a entire files-and-directory tree referenced by Root from memory. The procedure can also be used to eliminate unwanted parts of the tree.

Errors: None.

See also: ScanTree (32) BuildTree (32) , TreeBuildingTypes (29) procedure types (29) DirsToo (30)

The buildtree set of procedures don't have an example in the helpfile. See DirTest.pp in the XtdFPC package.

Chapter 5

EDos

EDOS, Dos functions not in unit DOS

EDos implements some simple functions typically used under Dos, like volumelabels, serial numbers, and drivetypedetections

This unit isn't portable at all (more portable stuf is moved to chapter 6

I removed all examples while TEX'ing the old examples, because I didn't like them. Almost all examples were incomplete, or too simple. I suggest you look at sysinfo or dfree for better fully working examples.

5.1 Types and variables

5.1.1 CMOSRec

Declaration:
```
TYPE CMOSRec= PACKED RECORD
           Sec,            Hr:Min:Sec current time. May deviate from time reported by
                           Windows or TIME command. Only synchronized on startup.
           AlSec,          AlHr:AlMin:AlSec time of next alarm interrupt. Mosttimes nonsense
           Min,
           AlMin,
           Hr,
           AlHr,
           DayOfWeek,      Day of week. Monday=1.
           Day,            Day/Month/Year (only 2 digits of year, other two are in century)
           Month,
           Year,
           RTCA,           Control registers of the Real Time Clock (RTC)
           RTCB,
           RTCC,
           RTCD,
           POSTStatus,     Diagnostic status byte
           ShutDownStatus, Shutdown status byte
           DiskType,       Floppy type  (lo(x)=A: Hi(X)=B:)
           Res1,
           HDType,         Hardddisk type (Lo(x)=HD1 Hi(X)=HD2) XT-only
           Res2,

           Equipment : BYTE; See sysinfo demo
```

```
BaseMem,          Basemem in kb
XTDMem    : WORD; Extended memory in kb (see alternate below)
HD_C,             modern harddisktypes (If Lo(CMOS.HDType)=15)
HD_D      : BYTE; modern harddisktypes (If Hi(CMOS.HDType)=15)
Res3      : ARRAY[0..18] OF BYTE;
Checksum,         (Sum of bytes 16 to 45) AND 65535
XTDMem2   : WORD; Extended memory in kb (see alternate above)
Century,          First 2 digits of year
Misc      : BYTE; Bit 7 set = top 128k installed,
                  bit 6 set = first user message?
Res4             : ARRAY[0..11] OF BYTE;
Res5             : ARRAY[0..63] OF BYTE;
END;
```

Description: This record is filled by GetCMOS (41) and contains the contents of the CMOS (the area where the BIOS keeps its data like harddisk time, amount of memory and the correct time.

Please beware that the contents of the first 10 fields (Sec..Year) and the century can be different from the actual BIOS content. This because GetCMOS (41) converts BCD to binary values.

Some information of the bit-level flag fields is included below.

- RTCA
  - Bit 7 = Update in progress
    - 0 = date & time can be read
    - 1 = time update busy
  - Bit 6-4 = Time frequency divider
    - 010 = 32.768 KHz
  - Bit 3-0 = Rate selection frequency
    - 0110 = 1.024 KHz sq. wve. freq.
- RTCB
  - Bit 7 = Clock update cycle
    - 0 = Update normally
    - 1 = Abort update in progress
  - Bit 6 = Periodic interrupt
    - 0 = disable (default), 1 = enable
  - Bit 5 = Alarm interrupt
    - 0 = disable (default), 1 = enable
  - Bit 4 = Update-ended interrupt
    - 0 = disable (default), 1 = enable
  - Bit 3 = Status register A sq. wve. freq.
    - 0 = disable (default), 1 = enable
  - Bit 2 = Date format
    - 0 = Calender in BCD format (default)
    - 1 = Calender in binary format
  - Bit 1 = 24-hour clock
    - 0 = 24-hour, 1 = 12-hour
  - Bit 0 = Daylight Savings Time
    - 0 = disable (default), 1 = enable

- RTCC
  - Bit 7 = IRQF flag
  - Bit 6 = PF Flag
  - Bit 5 = AF Flag
  - Bit 4 = UF Flag
  - Other bits reserved
- RTCD
  - Bit 7 = Valid CMOS RAM bit
    - 0 = battery dead, 1 = battery OK
  - Other bits reserved
- POST Diagnostic status
  - Bit 7 = Real-time clock power status
    - 0 = OK, 1 = not OK
  - Bit 6 = CMOS checksum status
    - 0 = good, 1 = bad
  - Bit 5 = POST config. status
    - 0 = valid, 1 = not valid
  - Bit 4 = POST Memory size check
    - 0 = OK, 1 = !OK
  - Bit 3 = Fixd disk/adapter init.
    - 0 = init OK, 1 = init bad
  - Bit 2 = CMOS time status
    - 0 = OK, 1 = !OK
  - Other bits reserved
- Shutdown code
  - 00h = Power on or soft reset
  - 04h = POST end; boot system
  - 05h = JMP dword ptr with EOI
  - 06h = Prot. mode tests OK
  - 07h = Prot. mode tests FAILED
  - 08h = Memory size FAILED
  - 09h = int 15h block move
  - 0Ah = JMP dword ptr with EOI
  - 0Bh = Used by 80386
- Installed equipment
  - Bits 7-6= Number of floppy drives
  - Bits 5-4= Primary display
    - 00= Adapter BIOS
    - 01= CGA 40 cols
    - 10= CGA 80 cols
    - 11= MDA
  - Bits 3-2= Reserved
  - Bit 1 = Math copro. present
  - Bit 0 = Floppy drive present
- POST information flag

–Bit 7 = Top 128K base memory status
  ∗0 = not installed
  ∗1 = installed
–Bit 6 = Setup program flag
  ∗0 = Normal (default)
  ∗1 = Output user message
–Other bits reserved

See also: GetCMOS (41)

5.1.2 TSwapInfo

Declaration:
```
TYPE PagerType = (Res0,                           Reserved
                  SW_NOPAGER,                     No paging system
                  SW_DOSPAGER,                    Paging through Dos
                  SW_IOSPAGER);                   Protected mode pager

     TSwapInfo     = RECORD                         GetSwapData record
                       FileName : String[255];    Location swapfile
                       FileSize : COMP;           Size swapfile
                       Pager    : PagerType;      Pager type, see above
                       END;
```

Description: This record is filled by GetSwapData (40) and contains the location and size of the swapfile, and the operating mode of the swapper (none,dos,protected)

See also: GetSwapData (40)

5.1.3 Drv

Declaration: `Drv :  LONGINT`

Description: A lot of procedures in this unit have a parameter called Drv.

This parameter identifies a drive by a positive number (table (5.1))

See also: a lot.

5.1.4 UART

Declaration: `TYPE UART = (uNoUART, uBadUART, u8250, u16450, u16550, u16550a);`

Description: The UART type as returned by TestUART (42). The enumeration names speak for themselves, except that NoUart can also be caused by the OS (e.g. Win95) blocking the detection.

See also: TestUART (42)

5.1.5 SubstExpand

Declaration: `VAR SubstExpand : BOOLEAN;`

Description: If this boolean is TRUE, TrueName (39), GetShortPathName (40) and GetLongPathName (40) will expand substed driveletters to their original paths.

See also: TrueName (39), GetShortPathName (40), GetLongPathName (40)

Table 5.1: Drv parameter table

Number	meaning
0	current (default) drive
1	drive a:
2	drive b: even if you don't have a second floppy drive
3	drive c:
4	drive d:
5	drive e:
6	drive f:
7	drive g:
8	drive h:
etc,	etc

5.2 Procedure and functions

5.2.1 GetSerial

Declaration: `FUNCTION GetSerial( Drv (37) : LONGINT) : LONGINT;`

Description: Returns the serialnumber as longint for drive Drv. (0=current, 1=A, 2=B etc)

Note: Only works for harddisks and floppies.

See also: Drv (37)

5.2.2 GetVolume

Declaration: `FUNCTION GetVolume( Drv (37) : LONGINT) : String;`

Description: Returns the volumename for drive Drv. (0=current, 1=A, 2=B etc)

Obtained with FindFirst (most compatible, other methods don't work on CDROMs, Zip drives and networks)

See also: Drv (37)

5.2.3 DriveType

Declaration: `FUNCTION DriveType( Drv (37) : LONGINT) : LONGINT;`

Description: Tries to identify the type of drive Drv. (0=current, 1=A, 2=B etc) table (5.2)

The procedure is quite efficient, and you can make your dosbased programs a lot more safe with it (no more invalid drive problems etc, or trying to mount cdroms without disks in it).

The procedure retrieves the number of floppy drives your system has (with NrFloppies (39)), and if that is 1, it assumes drive b: doesn't exist. This avoids renaming A: to B: as most modern mainboards do

It also uses standard procedure DiskFree to check all drives. This way removable drives (floppy,ZIP,Cdrom) without media loaded are non-existant, and NOT mounted.

The best way to detect the existance of drives is demonstrated in demo DFree or sysinfo.

Table 5.2: Returnvalues DriveType

ReturnValue	Drive type
0	Drive physically isn't available, or it's removable, and no disk is loaded
1	Remote (network, Ramdrive) disk drive
2	Fixed (hard) disk drive
3	Removable (floppy) disk drive
4	Substed drive
5	Cdrom

Note : The only system this goes wrong is when your bios declares two floppies in the biosdataarea, while you have only one.

See also: NrFloppies (39)

5.2.4 NrFloppies

Declaration: `FUNCTION NrFloppies:LONGINT`

Description: Retrieves the number of floppies from the BIOSdata area.

Some mainboards(most of them 486er) always return two even if only one diskdrive exists!

See also: NrFixedDisks (39)

5.2.5 NrFixedDisks

Declaration: `FUNCTION NrFixedDisks:LONGINT;`

Description: Retrieves the number of fixed disks from the BIOSdata area.

See also: NrFloppies (39)

5.2.6 LastDrv

Declaration: `FUNCTION LastDrv:LONGINT;`

Description: Returns highest valid logical drive

See also: NrFloppies (39) NrFixedDisks (39)

5.2.7 TrueName

Declaration: `PROCEDURE TrueName(VAR path :  String);`

Description: This procedure is similar to FExpand, but can also expands substituted drives to their real path.(Depending on SubstExpand (37))

This procedure uses a (formally) undocumented dosfunction, and you'd better use FExpand, unless you want to avoid substed drives for some reason, or distinguish between HD's,substed drives and cdroms.

Though undocumented, this dos function is used very often, and Dos 7.x contains a LongFileName version of it. It's generally considered to be secure.

See also: GetLongPathName (40)

5.2.8 GetLongPathName

Declaration: `PROCEDURE GetLongPathName (Short:  String; VAR Long:  String);`

Description: This procedure is basically a TrueName (39) which tries to return only long file names. (expands 8.3 notation with tildes to lfn when possible)

Depending on SubstExpand (37) substed drives are also expanded

See also: TrueName (39), GenerateShortName (41), GetShortPathName (40)

5.2.9 GetShortPathName

Declaration: `PROCEDURE GetShortPathName (Long:  String; VAR Short:  String);`

Description: This procedure is basically a TrueName (39) which only returns 8.3 (tilde-notation) filenames. This can be used to secure parameters which will be passed to non lfn-utils.

Depending on SubstExpand (37) substed drives are also expanded.

See also: TrueName (39), GenerateShortName (41), GetLongPathName (40)

5.2.10 ClusterSize

Declaration: `FUNCTION ClusterSize( Drv (37) :  LONGINT) : LONGINT;`

Description: Returns the clustersize (allocation size on harddisk). This is the smallest chunk that can be allocated on disk. (so a 1 byte files occupies clustersize bytes)

This procedure also works for FAT32 (it has LFN support), and is also corrected for CDROMs (Clustersize cdrom AFAIK always=2048)

See also: none.

5.2.11 NrDrives

Declaration: `FUNCTION NrDrives:LONGINT;`

Description: SHOULD return the number of logical drives, but Win95 always returns 32 AFAIK.

See also: NrFloppies (39)

5.2.12 GetSwapData

Declaration: `PROCEDURE GetSwapData(VAR Swp :     TSwapInfo (37));`

Description: Tries to locate the swapfile, and it's dimensions. If the call fails (No dos/win paging installed) the record is filled with CHR(0)

See also: TSwapInfo (37)

5.2.13 WinVer

Declaration: `Function WinVer:WORD;`

Description: Returns the windows version in BCD format (minor=LO(WinVer), major=HI(WinVer))

For Win95 it returns $400 (=4.00)

This can also be used to distinguish between dos 7.x dosmode and Win95 GUI.

See also: None.

5.2.14 GenerateShortName

Declaration: `PROCEDURE GenerateShortName(VAR Long: String; VAR Short: String);`

Description: Just like GetShortPathName, this procedure changes a longfilename to a short one. However GenerateShortName doesn't generate a tilde notation, but tries to fit as much characters as possible in the 8.3 space.

E.g. "Very long name.txt" becomes "verylong.txt" while GetShortPathName would do it like this: "verylo1̃.txt"

I don't use this procedure. I created it when trying to find something like GetShortPathName, and included it here, because it's ready.

See also: TrueName (39), GenerateShortName (41), GetShortPathName (40) GetLongPathName (40)

5.2.15 GetCMOS

Declaration: `PROCEDURE GetCMOS(VAR CMOS : CMOSRec (34));`

Description: Copies the first 64 bytes of the CMOS to the above packed CMOSRec (34) record

Before copying, GetCMOS tests bit 2 of the RTCB register if the date-time values of CMOS are coded as BCDs. If so, GetCMOS automatically performs a BCD2Binary conversion.

See also: CMOSRec (34)

5.2.16 Installed

Declaration: `FUNCTION Installed(Tsr:WORD):WORD;`

Description: Calls the multiplexer interrupt $2F (=47d).

```
IF Tsr \var{<} 255 THEN
  call \$2F with AH=Tsr, AL=0
 ELSE
  call \$2F with AX=Tsr BX=0;
```

The functionreturns values are shown in table (5.3)

Originally, multiplexvalues > 255 were errors(use (returnvalue AND 255)), however a lot of modern TSRs (like Windows) use the multiplex to return a version number in BCD format. Testing for values 0 and 1 is enough mosttimes, everything else is installed.

Some constants to use as parameter for this procedure are included in the interface section of EDos.

See also: None.

Table 5.3: Tsr Codes

Value	meaning
255	Tsr is installed
0	Tsr is not installed
1	Tsr not installed and not ok to install

5.2.17 TestUART

Declaration: `FUNCTION TestUART(Port : LONGINT): UART (37);`

Description: Tests if an UART (processor behind the comport) exists on port ”Port”, IOW does the comport corresponding with this address exist?. P.s. Windows sometimes blocks some ports. I can't detect COM1 under Windows. (The mouse is attached to it)

If I try to detect PS/2 ports under Windows, my soundcard hiks. Probably something doesn't decode the upper bits of the com-port adress :)

See also: Demo sysinfo

5.2.18 IsDevice

Declaration: `FUNCTION IsDevice(CONST Fnamex: String): BOOLEAN;`

Description: Returns TRUE if named file is actually a device. (Like CON,NUL, etc). Can be used to detect a device as input/output on commandline, or to verify existance of a certain devicedriver.

See also: None.

Chapter 6

The EFIO unit.

The EFIO unit is a kind of unit for miscellaneous routines that operate on files or filenames. As you can see, the number of routines is quite low. I do not define routines already existing in the RTL.

Also all directory scanning (routines that use FindFirst/FindNext) are located in EDirTree.

6.1 Types and constants

There is actually only one type in EFIO. This type is the return value of ArchiveMethod (43)

```
TYPE     ArchiveType = ( none,SQZ, ZIP, HPK, ZOO, LZH,
                         ARJ, DWC, ARC, PAK, A7P, HYP,
                         RAR,   Q, UC2, Gif, LBM, PCX,
                         WAV, BMP, MP3);
```

`About MP3` Please note that at the time of writing the MP3 file detection is a bit akward. It works, but I don't advise to run the detection on large amounts of binary files, since a lot binary files would (incorrectly) turn out to be MP3s. If you want to be sure about a file being a MP3, use the EMP3 unit to get the MP3-ID tag. If this call is successful, the file is almost certainly a MP3 file. However a lot of valid MP3's don't have a MP3-ID, so this method is also not fail safe.

6.2 Functions and procedures

6.2.1 ArchiveMethod

Declaration: `FUNCTION ArchiveMethod( FileName : String) : ArchiveType; (see section 6.1)`

Description: This function tries to identify the file-type of `FileName`. The function looks IN the file, it does not simply look at the extension. Also most detectionschemes are tested (in the Modula-2 version of XTDFPC), and quite secure. (see errors)

Errors: `MP3` see section 6.1

`DWC` The DWC-detection is not implemented, the value is only reserved for future use. `GIF, LBM, PCX, WAV, BMP` These detections work, but since most of these formats come in 20 different flavours additional detections will be necessary.

See also: section 6.1

```
Uses EFIO;

VAR FileName : String;

BEGIN
 FileName:='c:\xtdfpc18.zip';
 Write(Filename,' is of type : ');
 CASE ArchiveMethod(FileName) OF
    None :      Writeln('No archive type supported');
    SQZ  :      writeln('SQZ');
    ZIP  :      writeln('ZIP');
    HPK  :      writeln('HPK');
    ZOO  :      writeln('ZOO');
    LZH  :      writeln('LZH');
    ARJ  :      writeln('ARJ');
    DWC  :      writeln('DWC');      {Detection not working yet}
    ARC  :      writeln('ARC');
    PAK  :      writeln('PAK');
    A7P  :      writeln('A7P');
    HYP  :      writeln('HYP');
    RAR  :      writeln('RAR');
    Q    :      writeln('Q');
    UC2  :      writeln('UC2');
    GIF  :      writeln('GIF');
    LBM  :      writeln('LBM');
    PCX  :      writeln('PCX');
    WAV  :      writeln('WAV');
    BMP  :      writeln('BMP');
   ELSE
    Writeln('Unknown format (supported by Archivemthod, but not by this example)');
   END; {Case}
END.
```

6.2.2 FileExists

Declaration: `FUNCTION FileExists(FileName: String): Boolean;`

Description: This boolean function returns True if the file `FileName` exists, else it returns False. Closes the file if it exists, principe copied from the BP7 help.

Errors: Seems not to work for a directory. (That requires a FileExists based on FindFirst FindNext, instead of trying to open the file)

```
Uses EFIO;

BEGIN
```

```
  IF FileExists('C:\autoexec.bat') THEN
   Writeln('This is probably a msdos system')
  ELSE
   Writeln('This is probably no msdos system');
END.
```

6.2.3 WrHex

Declaration: `PROCEDURE WrHex (InValue:WORD);`
`PROCEDURE WrHex (VAR F : Text;InValue:WORD);`

Description: This procedure translates the binary value `InValue` to a 4 characters hex representation, and outputs it to stdout or file `F`.

Errors: None, but in Delphi mode HexStr exists which does the same.

See also: WrBinary (46) WrLngBinary (46) WrLngHex (45) WrOct (46) WrLngOct (46)

```
uses EFIO;

VAR Value16  : WORD;
    Value32  : LONGINT;

BEGIN
  Value16:=12345;              { Define a 16 bits value}
  Value32:=1234567890;             { Define a 32 bits value}

  Write(Value16, ' decimal = $'); WrHex(value16); Write(' hexadecimal, ');
  WrOct(Value16); Write('o octal and %'); WrBinary(value16);writeln(' binary.');

  Write(Value32, ' decimal = $'); WrLngHex(value32); Write(' hexadecimal, ');
  WrLngOct(Value32); Writeln('o octal');
  Write(' and %'); WrLngBinary(value32);writeln(' binary.');
END.
```

6.2.4 WrLngHex

Declaration: `PROCEDURE WrLngHex (InValue:CARDINAL);`
`PROCEDURE WrLngHex (VAR F:Text;InValue:CARDINAL);`

Description: This procedure translates the binary value `InValue` to a 8 character hex representation, and outputs it to stdout or file `F`.

Errors: None, but in Delphi mode HexStr exists which does the same.

See also: WrBinary (46) WrLngBinary (46) WrHex (45) WrOct (46) WrLngOct (46)

For an example see WrHex (45)

6.2.5 WrOct

Declaration: `PROCEDURE WrOct (InValue:WORD);`
`PROCEDURE WrOct (VAR F : Text;InValue:WORD);`

Description: This procedure translates the binary value `InValue` to a 6 characters octal representation, and outputs it to stdout or file `F`.

Errors: None

See also: WrBinary (46) WrLngBinary (46) WrHex (45) WrLngHex (45) WrLngOct (46)

For an example see WrHex (45)

6.2.6 WrLngOct

Declaration: `PROCEDURE WrLngOct (InValue:CARDINAL);`
`PROCEDURE WrLngOct (VAR F:Text;InValue:CARDINAL);`

Description: This procedure translates the binary value `InValue` to an 11 character octal representation, and outputs it to stdout or file `F`.

Errors: None

See also: WrBinary (46) WrLngBinary (46) WrHex (45) WrLngHex (45) WrOct (46)

For an example see WrHex (45)

6.2.7 WrBinary

Declaration: `PROCEDURE WrBinary (InValue:WORD);`
`PROCEDURE WrBinary (VAR F : Text;InValue:WORD);`

Description: This procedure translates the binary value `InValue` to a 16 characters binary representation, and outputs it to stdout or file `F`.

Errors: None, but in Delphi mode BinStr exists which does the same.

See also: WrLngBinary (46) WrHex (45) WrLngHex (45) WrOct (46) WrLngOct (46)

For an example see WrHex (45)

6.2.8 WrLngBinary

Declaration: `PROCEDURE WrLngBinary (InValue:CARDINAL);`
`PROCEDURE WrLngBinary (VAR F:Text;InValue:CARDINAL);`

Description: This procedure translates the binary value `InValue` to an 32 character binary representation, and outputs it to stdout or file `F`.

Errors: None, but in Delphi mode BinStr exists which does the same.

See also: WrBinary (46) WrHex (45) WrLngHex (45) WrOct (46) WrLngOct (46)

For an example see WrHex (45)

6.2.9 ExtensionPos

Declaration: `FUNCTION ExtensionPos ( CONST s : String) : WORD;`

Description: Returns the position of the extension in path `s`, or MAX(WORD) (=65535) if no extension was found. Actually this function was internal, but the function can be used to check if extension exists, so I moved it to the interface.

Errors: None.

See also: ChangeExtension (48) RemoveExtension (47) AddExtension (47)

```
Uses EFIO;

VAR Name, Ext : String;

BEGIN
 Name:='Hello.tar.gz';
 Writeln('Original Name : ',Name);
 RemoveExtension(Name);
 ChangeExtension(Name,'tgz');
 Writeln('New name  : ',Name);
 RemoveExtension(Name);
 AddExtension(Name,'tar.bz2');
 Writeln('Repacked name : ',Name);
 Writeln;
 Write('Let'#39's determine the basename of ',Name);
 WHILE ExtensionPos(Name)<>65535 DO RemoveExtension(Name);

 Writeln('   Base name: "',Name,'"');

END.
```

6.2.10 RemoveExtension

Declaration: `PROCEDURE RemoveExtension ( VAR s : String) ;`

Description: Calls ExtensionPos, and deletes the (last) extension if it exists.

Errors: None.

See also: ChangeExtension (48) ExtensionPos (47) AddExtension (47)

For an example see ExtensionPos (47)

6.2.11 AddExtension

Declaration: `PROCEDURE AddExtension ( VAR s : String; CONST Extension : String) ;`

Description: Under Linux or when LfnSupport=TRUE : Add (another) extension to `S`. If LfnSupport=FALSE : Only add an extension to `S` of none present.

Errors: None, but note different behaviour depending on LfnSupport.

See also: RemoveExtension (47) ExtensionPos (47) ChangeExtension (48)

For an example see ExtensionPos (47)

6.2.12 ChangeExtension

Declaration: `PROCEDURE ChangeExtension ( VAR s : String; CONST Extension : String) ;`

Description: Changes the (last) extension of the filename in `S` to `extension`. If no extension exists, the extension is added.

Errors: None.

See also: RemoveExtension (47) ExtensionPos (47) AddExtension (47)

For an example see ExtensionPos (47)

6.2.13 WrStrAdj

Declaration: `PROCEDURE WrStrAdj(CONST InS: String;L : LONGINT);`
`PROCEDURE WrStrAdj(VAR F: Text;CONST InS: String;L : LONGINT);`

Description: This procedure is roughly the same as write(Ins:L) but pads on the other side if L is smaller than zero.

Errors: None.

```
Uses EFIO;

BEGIN
 Writeln('012345678901234567890123456789');
 WriteLn('"','text':25,'"');
 Write('"');  WrStrAdj('text',25);  Writeln('"');
 Write('"');  WrStrAdj('text',-25);  Writeln('"');

END.
```

6.2.14 Touch

Declaration: `PROCEDURE Touch(Const FileName:String);`

Description: Simply the well known touch command (Set filetimedate of file(s) to current datetime). Directories and wildcards supported. NOT recursive. Use EDirTree or ODirTree to run a procedure like this recursive.

Errors: None.

No example yet.

6.2.15 DelDir

Declaration: `PROCEDURE Deldir(Dir : PathStr);`

Description: `CAUTION, DANGERUOUS` Recursively removes all contents of DIR, hidden files and directories inclusive, like MsDos Deltree, or Linux rm -rf.

Errors: Hasn't been thoroughly tested on Linux or Win32. Should be safe, but test carefully. (e.g. on substed drive)

6.2.16 FileAppend

Declaration: `PROCEDURE FileAppend(VAR F :Text; CONST FileName:String);`

Description: Performs Assign (F,S); Append(F); but creates the file if it doesn't exists.

Errors: None.

```
uses EFIO;

VAR F : Text;

CONST Filename='test.txt';

BEGIN
  Assign(F,Filename);
  Rewrite(F);
  Writeln(F,'operation one');
  Close(F);
  FileAppend(F,Filename);
  Writeln(F,'Operation two');
  Close(F);
END.
```

6.2.17 MkFullDir

Declaration: `PROCEDURE MkFullDir(CONST InPath:String);`

Description: Create a directory, but the procedure also works for 'd:
prog
compilers
pp' if 'd:
prog' doesn't exist.

Errors: Go32V2 : The drive(if specified) must exist though.

No example yet.

6.2.18 WrArrChar

Declaration: `PROCEDURE WrArrChar(Data :  PChar; sizedata:  LONGINT);`

Description: This procedure outputs the first `sizedata` bytes from `Data` to screen, replacing CHR(0) and CHR(13) by a linefeed (using writeln).

The procedure was created because of the fact that FPC can't have textual constants longer than 255. I created a workaround (see ../devel/data2inc.pp), and this procedure is the output part of it. Most of my demos use this procedure to write their "usage-screen" to stdout.

Errors: None, but be aware that Data is not an ordinary PCHAR! SizeData bytes will be printed, regardless of CHR(0)'s in Data!

No example yet, see demos like crtolf and indexer.

Chapter 7

ELib

ELib used to be the biggest unit by far (100+ procedures) in the Modula2 version, because it was a kind of miscellaneous assembler unit

In Pascal it's a lot smaller, and contains a few OSdependant and platform dependant routines. Mainly primitives for lowlevel and interfacing procedures.

Some of the functions now have a RTL equivalent. When I do the next face-lift of this unit, those procedure will dissappear

7.1 Types

7.1.1 CHARSET

Declaration: `TYPE CHARSET = SET OF CHAR;`

Description: Defining this simple type saves a lot of trouble in Modula-2. This is a bit less important in Pascal (because you can define set of chars constants without a type identifier). I however think it's cleaner, and also EPasStr routines depend on it. Also you base your procedures on a identifier which you can change without redefining a compiler type. This can be handy for 16-bits character types, though that is not really planned for now.

See also: chapter 9

7.2 Procedures and Functions

7.2.1 FillCard

Declaration: `PROCEDURE FillCard(var x;count :  LONGINT;value :  cardinal);`

Description: FillChar, FillWord, and.....

Yes, FillCard. Fill memoryblock starting on address `X` with `Count` times value. So a block 4*`Count` will be filled because CARDINAL is 4 bytes wide. Can also be used for filling with a longint, but you'll need an (implicit) typecast for that.

See also: none.

7.2.2 ScanR

Declaration: `FUNCTION ScanR(VAR Adr;Value :  BYTE;Count:  LONGINT):LONGINT;`

Description: Search for byte `Value` in memory block starting on address `adr`, for maximal `Count` bytes. Returns zerobased offset with respect to `adr`, or 1 when Value isn't found.

In fact this is a "rep scasb" instruction with Pascal header and some instructions that handle the notfound case.

See also: None.

7.2.3 ISqrt

Declaration: `FUNCTION ISqrt(Indata:CARDINAL):CARDINAL;`

Description: Equivalent to ISqrt:=Trunc(SQR(Float(InData)));

Square root of `Indata` rounded down. Entirely integer, no reals used. Old 386sx trick, and my first 32-bits code :-) Probably a lot slower than copro today, since it uses a loop. Assembler FPUroot is one instruction :). Well, nostalgia it is then.

7.2.4 GetKey

Declaration: `FUNCTION GetKey:WORD;`

Description: This is the oldest procedure in entire XTDLIB. It's a shell to the Crt `ReadKey` procedure, which avoids problems with function keys. Function keys (like F1) are returned to ReadKey as two characters, the first being zero. GetKey simply calls ReadKey, and if ReadKey is zero, it calls ReadKey again and returns the second readkey SHL 8.

See also: Keys.inc contains some standard values for GetKey

7.2.5 SetCursorSize

Declaration: `PROCEDURE SetCursorSize(A:WORD);`

Description: BIOS function, so Go32V2 (and maybe Go32V1) only. BP equivalent implemented in BPGo32.pas

Set cursorsize, the high byte is the first scanline, the low byte the last (lowest) scanline. A form of this procedure is included in Crt (SetCurSize($090A) is the same as Crt.CursorOn, and SetCurSize($FFFF) as CursorOff) but not exported.

chapter 10 uses this procedure and GetCursorSize (51) to save and restore cursorsettings when changing windows.

Another application is to save deviating cursorshapes before shelling to dos, and restore afterwards.

See also: GetCursorSize (51)

7.2.6 GetCursorSize

Declaration: `FUNCTION GetCursorSize:WORD;`

Description: BIOS function, so Go32V2 (and maybe Go32V1) only. BP equivalent implemented in BPGo32.pas

Stores the cursorshape in a word. The high byte is the first scanline, the low byte the last (lowest) scanline.

chapter 10 uses this procedure and SetCursorSize (51) to save and restore cursorsettings when changing windows.

Another application is to save deviating cursorshapes before shelling to dos, and restore afterwards

See also: SetCursorSize (51)

7.2.7 set_fs_to_dosmem

Declaration: `PROCEDURE set_fs_to_dosmem;`

Description: (Go32V2) only

This procedure reloads %fs with DOSMEMSELECTOR, which it should be. Can be handy to do this after very lowlevel and/or odd assembler or external routines, to reset %fs to point to dos real mode memory.

See also: None.

Chapter 8

The EMP3 unit.

This unit DOES NOT play MP3's or anything like that. It's only a check for the MP3filetype, plus the reading/writing of the standard MP3-tag.

The EMP3 unit has emerged from a failed effort to update ArchiveMethod (43) to recognize MP3 files. The detection was different from the others, and more complicated. Since meanwhile I also implementing reading and writing tags, I decided to give MP3 its own unit.

The main demonstration program for this unit is File2Tag, which converts the filename (and path), to an mp3tag.

8.1 How the MP3 Check is implemented

The detection of MP3 files is somewhat difficult. It's based on finding a byte $FF (255 decimal), and then a byte with the highnibble set to $F.

However this doesn't necessarily has to exist at the beginning of a file, or even within the first 5kb of a file. The detectionroutine (IsMp3 (56)) of this unit checks for the sequence $FF $Fx with x not equal to $F in the first 12kb. The reason for not allowing $FF, $FF is because some valid MP3 files with some header prefixed have this value in their header. Adding this restrain increased reliability of the check.

I think for the detection to be better, I need to now more about the format. I guess some of the bytes after the first signature indicate the number of bytes to the next signature. This isn't implemented.

I tested on some random files. If you run this detection routine on random binary files you will probably find a nice percentage non-mp3 files detected as mp3-files. So I suggest you only test on files which ARE probably mp3 files (extension .mp3). If you do so, the IsMp3 (56) is quite reliable again. All files that ARE mp3 files, but not detected as such, are not standard MP3 files. Probably most mp3players, maybe except the major MP3 players (read WinAmp) won'y read them. I (and a friend) tested 4000 files, and less than 0.1failed the test. Anything that improves the detection is welcome.

8.2 Types and constants

8.2.1 Genre byte

The MP3 tag provides one entire byte to store some genre information. As far as I know, the first 80 (0..79) ordinal values are predefined.

I defined constants for all values, and even an array with a short (10 character) textual description of each value in the includefile genretag.inc.

8.2.2 Genre

Declaration: `CONST Genre :  ARRAY[0..79] OF String[10]= (declarations)`

Description: "Genre" is an array [0..79] of string[10], filled with a short textual description of each corresponding genre ordinal (subsection **??**)

See also: subsection 54 ID3 constants (54)

8.2.3 ID3 constants

Declaration: `CONST ID3_somegenre=somevalue`

Description: The ordinal values of the genres as constants.

See also: subsection 54 Genre (54)

8.2.4 Gettag errorcodes

Declaration:
```
CONST MP3_TAG_ALLOK = 0; MP3_TAG_NOT_FOUND = 1; Returned by GetTag when
no tag was found MP3_TAG_GET_ERROR = 2; Returned by GetTag when retrieving
tag failed (probably file too small) MP3_TAG_PUT_ERROR = 3; Returned by
SetTag when saving tag failed (probably disk full or file/network write
protected)
```

Description: These codes are returned by GetTag (55) to indicate success or failure

See also: GetTag (55), SetTag (55)

8.2.5 BitRates

Declaration:
```
CONST BitRates :  ARRAY [1..2,1..3,1..15] OF INTEGER= ( ((0,32,64,96,128,160,192,224,256,288,32
(0,32,48,56, 64, 80, 96,112,128,160,192,224,256,320,384), (0,32,40,48,
56, 64, 80, 96,112,128,160,192,224,256,320)), ((0,32,48,56, 64, 80, 96,112,128,144,160,176,192,2
(0, 8,16,24, 32, 40, 48, 56, 64, 80, 96,112,128,144,160), (0, 8,16,24,
32, 40, 48, 56, 64, 80, 96,112,128,144,160)) );
```

Description: This array returns the bitrate if you know Mpeg type, layer and an ordinal for bitrate.

The indexes are

$$mpeg, layer, speedcode$$

. The speedcode is found by the following formula: (Ident AND $F00000) SHR 20. Ident is the returncode of IsMp3 (56) when it's not 1. (which means no valid MP3). (see sourcecode of DumpIndentifier (**??**) for an example of what information you can extract out of a Ident returnvalue)

See also: IsMp3 (56), DumpIndentifier (**??**) SampleFreq (55)

8.2.6 SampleFreq

Declaration: `CONST SampleFreq : ARRAY[1..2,0..2] OF WORD= (( 44100 , 48000 , 32000), ( 22050 , 24000 , 16000));`

Description: This array returns the samplefrequency Mpeg type and an index for the frequency.

The indexes are [(mpeg-1) XOR 1,freqcode]. The freqcode is found by the following formula: (Ident AND $C0000) SHR 18. Ident is the returncode of IsMp3 (56) when it's not 1. (1 means no valid MP3). (see sourcecode of DumpIndentifier (??) for an example of what information you can extract out of a Ident returnvalue)

See also: IsMp3 (56), DumpIndentifier (??), BitRate (??)

8.2.7 ID3Tag

Declaration: `type ID3TAG = Record Used for easy update'ing, tags are internally array of char Songname : String[30]; Artist : String[30]; Album : String[30]; Year : String[4]; Comment : String[30]; GenreID : Byte; end;`

Description: This records represents a MP3tag. It can be read and written by using GetTag (55) and SetTag (55).

The record is not binary compatible with the actual mp3tag. (which is array of char style, packed, and has an identifier field), but the translation is done by the procedure that read and write the tag. I just mention it, so that you won't try to write it directly :)

See also: SetTag (55), GetTag (55)

8.3 Procedure and functions

8.3.1 GetTag

Declaration: `FUNCTION GetTag(CONST Filename:String;VAR Tag : ID3Tag ):LONGINT;`

Description: Checks if the filename has a valid MP3 (ID3)tag, and reads it or returns a code indicating an error took place (Gettag errorcodes (54)).

If the procedure finds a valid tag, it converts the tag to a bit more usable record (pascal string instead of padded array of char).

See also: SetTag (55) ID3Tag (55), Gettag errorcodes (54), DumpTag (56)

8.3.2 SetTag

Declaration: `FUNCTION SetTag(CONST Filename:String;CONST Tag : ID3Tag ):LONGINT;`

Description: Writes the ID3tag to filename `FileName`. If an tag is already present, the tag is replaced by the `Tag`, otherwise `Tag` is appended.

See also: GetTag (55) ID3Tag (55), Gettag errorcodes (54), DumpTag (56)

8.3.3 IsMp3

Declaration: `FUNCTION IsMp3(FileName:String):LONGINT;`

Description: Detects if `filename` is a MP3 file. If yes, than return 32bit identifier as a LONGINT (e.g. FF FE 01 02 becomes $0201FEFF) If it isn't, return 1

See DumpIndentifier (??) for what you can do with the returnvalue.

See also: DumpIndentifier (??)

8.3.4 DumpTag

Declaration: `Procedure DumpTag(Tag:ID3Tag);`

Description: Prints the value of a mp3tag record to screen. More a debug procedure, but can be useful.

See also: GetTag (55), SetTag (55), ID3Tag (55)

8.3.5 DumpIdentifier

Declaration: `Procedure DumpIdentifier(Ident:LONGINT);`

Description: This procedure analyses the returnvalue of IsMp3 (56) and displays some of the values (using BitRates (54) and SampleFreq (55)

The use of this procedure is quite little, except for debug purposes, but you can look at the source, and see how it obtains its data.

See also: GetTag (55), SetTag (55), ID3Tag (55)

Chapter 9

The EPasStr unit.

This is the documentation for `EPASSTR` unit which contains the pascal and ANSI string routines of XTDFPC.

The main target is FPC (Linux and Go32V2 tested), though the non assembler routines will work with BP. (the generic include file of XTDFPC will automatically turn off the assembler when epasstr is compiled for BP)

Please note that right now, the AnsiString routines are alpha, and only sparsely tested.

9.1 Functions and procedures.

9.1.1 LTrim

Declaration: `PROCEDURE LTrim (VAR P : String;Ch:CHAR);`
`PROCEDURE LTrim (VAR P : AnsiString;Ch:CHAR);`

Description: Strips all characters Ch from the left (beginning) of the string P.

Errors: Pascal string : None, AnsiString: Untested

See also: RTrim (57), KillChar (58), KillBChar (58), KillChrTot (59), StripChar (59)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='        text';
 LTrim(P,' ');
 Writeln(P);            {writes  'text'}
END.
```

9.1.2 RTrim

Declaration: `PROCEDURE RTrim(VAR P:String;Ch:Char);`
`PROCEDURE RTrim(VAR P:AnsiString;Ch:Char);`

Description: Strips all characters `Ch` from the right (the end) of the string `P`.

Errors: Pascal string : None, AnsiString: Untested

See also: LTrim (57), KillChar (58), KillBChar (58), KillChrTot (59), StripChar (59)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text      ';
 RTrim(P,' ');
 Writeln(P);             {writes 'text'}
END.
```

9.1.3 KillChar

Declaration: `PROCEDURE KillChar(VAR S : STRING;CONST CSet:CHARSET);`
`PROCEDURE KillChar(VAR S : AnsiString;CONST CSet:CHARSET);`

Description: LTrim (57) but then for an entire character set. Strips all characters in set `CSet` from the begin (left) of string `P`.

Errors: Pascal string : None, AnsiString: Untested

See also: LTrim (57), RTrim (57), KillBChar (58), KillChrTot (59), StripChar (59)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='A B A B A Btext';
 KillChar(P,['A','B',' ']);
 Writeln(P);                     {writes 'text'}
END.
```

9.1.4 KillBChar

Declaration: `PROCEDURE KillBChar(VAR S : String;CONST CSet:CHARSET);`
`PROCEDURE KillBChar(VAR S : AnsiString;CONST CSet:CHARSET);`

Description: (RTrim (57) but then for an entire character set). Strips all characters in set `CSet` from the end (right) of string `P`.

Errors: Pascal string : None, AnsiString: Untested

See also: LTrim (57), RTrim (57), KillChar (58), KillChrTot (59), StripChar (59)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='textA B A B A B';
 KillBChar(P,['A','B',' ']);
 Writeln(P);             {writes 'text'}
END.
```

9.1.5 StripChar

Declaration: `PROCEDURE StripChar(VAR S:String;C:CHAR);`
`PROCEDURE StripChar(VAR S:AnsiString;C:CHAR);`

Description: Strips all characters `C` from string `S`.

Errors: Pascal string : None, AnsiString: Untested

See also: LTrim (57), RTrim (57), KillChar (58), KillBChar (58), KillChrTot (59)

```
Uses EPasStr;

VAR P : string;

BEGIN
 P:='text\ A A A A A ';
 StripChar(P,'A');
 Writeln(P);          {writes  'text\       '}
END.
```

9.1.6 KillChrTot

Declaration: `PROCEDURE KillChrTot(VAR S : String;CONST CSet:CHARSET);`
`PROCEDURE KillChrTot(VAR S : AnsiString;CONST CSet:CHARSET);`

Description: Strips all characters in set CSet from string S.

Errors: Pascal string : None, AnsiString: Untested

See also: LTrim (57), RTrim (57), KillChar (58), KillBChar (58), StripChar (59)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text\ A A A A A ';
 KillChrTot(P,['A',' ']);
 Writeln(P);          {writes  'text\'}
END.
```

9.1.7 AppendBackSlash

Declaration: `PROCEDURE AppendBackslash(VAR S : String);`
`PROCEDURE AppendBackslash(VAR S : AnsiString);`

Description: Appends a backslash ('\') to the end of string `S` if it's not already there. Under Linux it appends a '/'. Used as a primitive for programs which create a lot of paths.

Using this procedure makes programs more safe. The Dos rtl procedures (the LFN ones anyway) don't work right on paths with two backslashes in it, probably because of the UNC (\\server\share\) notation of networkdrives. Using (P)AppendBackslash avoids such problems because it doesn't append a backslash if it's already there, like S:='S'+'\'+name; would.

Errors: Pascal string : None, AnsiString: Untested

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text\';
 AppendBackslash(P);
 Writeln(P);          {writes 'text\'}
 P:='text';
 AppendBackslash(P);
 Writeln(P);          {writes 'text\'}
END.
```

9.1.8 ReplaceChar

Declaration: `PROCEDURE ReplaceChar(VAR S : String;ReplaceMe,ReplaceWith:CHAR);`
`PROCEDURE ReplaceChar(VAR S : AnsiString;ReplaceMe,ReplaceWith:CHAR);`

Description: Replace in string the character `ReplaceMe` with `RepWith`

Errors: Pascal string : None, AnsiString: Untested

See also: ExpandTabs (69), CompressTabs (69)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text\ A A A A A ';
 ReplaceChar(P,'A','B');
 Writeln(P);                    {writes 'text\ B B B B B '}
END.
```

9.1.9 CharPos

Declaration: `FUNCTION CharPos(CONST S :String;C:Char):LONGINT;`
`FUNCTION CharPos(CONST S :AnsiString;C:Char):LONGINT;`

Description: `Pos` (f) or one char (`C`) only. Faster than an ordinary Pos, returns 0 when character not found, just like ordinary Pos.

CharPos starts searching at the beginning of the string.

Errors: Pascal string : None, AnsiString: Untested

See also: NextCharPos (61), RCharPos (61), NextRCharPos (62), CharPosSet (62), NextCharPosSet (62)

```
Uses EPasStr;

VAR P : String;
```

```
BEGIN
 P:='text\ A A A A A ';
 Writeln(CharPos(P,'A'));   {writes 7 }
END.
```

9.1.10 NextCharPos

Declaration: `FUNCTION NextCharPos(CONST S:String;C:Char;Count:LONGINT):LONGINT;`
`FUNCTION NextCharPos(CONST S:AnsiString;C:Char;Count:LONGINT):LONGINT;`

Description: seemPos for one char only. Faster than an ordinary Pos. This particular version starts searching string `S` at character number `count`, and searches for `C` towards the end of the string. The function returns a standard index in the string (1..Length(S), or 0 when not found.

CharPos (60) starts searching at the beginning of the string.

Errors: Pascal string : None, AnsiString: Untested

See also: CharPos (60), RCharPos (61), NextRCharPos (62), CharPosSet (62), NextCharPosSet (62)

```
Uses EpasStr;

VAR P : String;

BEGIN
 P:='text\ A A A A A ';
 Writeln(NextCharPos(P,'A',8));   {writes 9}
END.
```

9.1.11 RCharPos

Declaration: `FUNCTION RCharPos(CONST S :String;C:Char):LONGINT;`
`FUNCTION RCharPos(CONST S :AnsiString;C:Char):LONGINT;`

Description: `Pos` (f) or one char only. Faster than an ordinary Pos, returns 0 when not found. This version starts searching for `C` at the back of the string `S`, back to the beginning. The function returns a standard index in the string. (1..Length(S), or 0 if character `C` was not found).¡p¿

Errors: Pascal string : None, AnsiString: Untested

See also: CharPos (60), NextCharPos (61), NextRCharPos (62), CharPosSet (62), NextCharPosSet (62)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text\ A A A A A ';
 Writeln(RCharPos(P,'A'));   {writes 15 }
END.
```

9.1.12 NextRCharPos

Declaration: `FUNCTION NextRCharPos(CONST S:String;C:Char;Count:LONGINT):LONGINT;`
`FUNCTION NextRCharPos(CONST S:AnsiString;C:Char;Count:LONGINT):LONGINT;`

Description: `Pos` (f) or one char only. Faster than an ordinary Pos, returns 0 when character `C` was not found. This version starts searching for `C` at the position Count (1..Length(S)) back to the beginning (1) of the string `S`. The function returns a standard index in the string. (1..Length(S), or 0 if character `C` was not found).¡p¿

Errors: Pascal string : None, AnsiString: Untested

See also: CharPos (60), NextCharPos (61), RCharPos (61), CharPosSet (62), NextCharPosSet (62)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text\ A A A A A ';
 Writeln(NextRCharPos(P,'A',14));   {writes 13 (the last but one A)}
END.
```

9.1.13 CharPosSet

Declaration: `FUNCTION CharPosSet(CONST S : String;CONST CSet:CHARSET):LONGINT;`
`FUNCTION CharPosSet(CONST S : AnsiString;CONST CSet:CHARSET):LONGINT;`

Description: Returns the first occurance in string S of a character in charset CSet, returns 0 when no matching character was found, the position of the character (1..Length(S)) otherwise.

Errors: Pascal string : None, AnsiString: Untested

See also: CharPos (60), NextCharPos (61), RCharPos (61), NextRCharPos (62), NextCharPosSet (62)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text\ A A A A A ';
 Writeln(CharPosSet(P,['A','\']));   {writes 5 }
END.
```

9.1.14 NextCharPosSet

Declaration: `FUNCTION NextCharPosSet(CONST S : String;CONST C:CHARSET;Count:LONGINT):LONGINT;`
`FUNCTION NextCharPosSet(CONST S : AnsiString;CONST C:CHARSET;Count:LONGINT):LONGINT;`

Description: Returns the (next) occurance in string `S` of a character in charset `CSet`, while starting on positionz returns 0 when no matching character was found, the position of the character (1..Length(S)) otherwise.

Errors: Pascal string : None, AnsiString: Untested

See also: CharPos (60), NextCharPos (61), RCharPos (61), NextRCharPos (62), NextCharPosSet (62)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='text\ A B A A A ';
 Writeln(NextCharPosSet(P,['A','B'],8));  {writes 9}
END.
```

9.1.15 StripDoubleChar

Declaration: `PROCEDURE StripDoubleChar(VAR S:String;C:Char);`
`PROCEDURE StripDoubleChar(VAR S:AnsiString;C:Char);`

Description: If 2 or more sequential 'C' chars exist in string, strip all but one ' 1 2 3 5 6 ' becomes ' 1 2 3 5 6 ' Used to make mail from Fido-newbies readable (run it for ' ','.' and '!') :-)

Errors: Pascal string : None, AnsiString: Untested

See also: ReplaceChar (60)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:=' 1 2  3   4    5';
 StripDoubleChar(P,' ');
 Writeln(P);              { Writes ' 1 2 3 4 5'}
END.
```

9.1.16 LowerCase

Declaration: `PROCEDURE LowerCase(VAR S : String);`
`PROCEDURE LowerCase(VAR S : AnsiStrng);`

Description: All (normal, not international ones) characters lowercase

Errors: Pascal string : None, AnsiString: Untested

See also: UpperCase (64)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='ABCDE';
 LowerCase(P);
 Writeln(P);            {writes 'abcde'}
END.
```

9.1.17 UpperCase

Declaration: `PROCEDURE UpperCase(VAR S : String);`
`PROCEDURE UpperCase(VAR S : AnsiStrng);`

Description: All (normal, not international ones) characters lowercase

Errors: Pascal string : None, AnsiString: Untested

See also: LowerCase (63)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='abcde';
 UpperCase(P);
 Writeln(P);          {writes 'ABCDE'}
END.
```

9.1.18 StrToBinary

Declaration: `FUNCTION StrToBinary(CONST S : String;Bits :  CARDINAL):CARDINAL;`
`FUNCTION StrToBinary(CONST S : AnsiString;Bits :  CARDINAL):CARDINAL;`

Description: Get first `Bits` bits or digits from `S` (like S:='0101010'), and return their binary value. Allowable range `Bits` range is 0..32, though 0 is useless (returns 0)

Errors: Pascal string : None, AnsiString: Untested

See also: StrToOct (64) StrToHex (65) OctToStr (65) HexToStr (66) BinaryToStr (65)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='010101010101010';
 Writeln(StrToBinary(P,Length(P))); {writes '10922'}
 P:='666';
 Writeln(StrToOct(P,Length(P))); {writes '438'}
 P:='800';
 Writeln(StrToHex(P,Length(P))); {writes '2048'}
END.
```

9.1.19 StrToOct

Declaration: `FUNCTION StrToOct (CONST S : String;Digits:  CARDINAL):CARDINAL;`
`FUNCTION StrToOct (CONST S : AnsiString;Digits:  CARDINAL):CARDINAL;`

Description: Get first `Digits` octal digits from `S` (like S:='766'), and return their binary value. (766o=502d) Allowable range `Digits` range is 0..11, (=8log($2\hat{3}2$) rounded upward) though 0 is useless (returns 0).

Errors: Pascal string : None, AnsiString: Untested

See also: StrToBinary (64) StrToHex (65) OctToStr (65) HexToStr (66) BinaryToStr (65)

For an example, see StrToBinary (64)

9.1.20 StrToHex

Declaration: `FUNCTION StrToHex (CONST S : String;Digits: CARDINAL):CARDINAL;`
`FUNCTION StrToHex (CONST S : AnsiString;Digits: CARDINAL):CARDINAL;`

Description: Get first `Digits` hex digits from `S` (like S:='8FA'), and returns their binary value. Allowable range `Digits` range is 0..8, though 0 is useless (returns 0).

Errors: Pascal string : None, AnsiString: Untested

See also: StrToBinary (64) StrToOct (64) OctToStr (65) HexToStr (66) BinaryToStr (65)

For an example, see StrToBinary (64)

9.1.21 BinaryToStr

Declaration: `PROCEDURE BinaryToStr(VAR S : String;Value,Bits : CARDINAL);`
`PROCEDURE BinaryToStr(VAR S : AnsiString;Value,Bits : CARDINAL);`

Description: Convert `Value` to a binary representation and write it to `S`, the procedure always writes `Bits` digits/bits. If you specify more bits (Like in BinaryToStr(S,1,10)), the result will be left padded with zeroes. If you specify `Bits` less than the number of bits needed to represent `Value`, only the rightmost `Bits` digits will be written. (like `Value` was ANDed with 2^`Bits`-1)

The `Bits` parameter can lie in the range 0..32, but 0 does nothing.

Errors: Pascal string : None, AnsiString: Untested

See also: StrToBinary (64) StrToOct (64) StrToHex (65) BinaryToStr (65) HexToStr (66)

```
Uses EPasStr;

VAR S   : String;

BEGIN
  BinaryToStr(S,$2AAA,16);
  Writeln($2AAA,' = ',S,'b');
  OctToStr(S,$2AAA,6);
  Writeln($2AAA,' = ',S,'o');
  HexToStr(S,$2AAA,4);
  Writeln($2AAA,' = ',S,'h');
END.
```

9.1.22 OctToStr

Declaration: `PROCEDURE OctToStr (VAR S : String;Value,Digits : CARDINAL);`
`PROCEDURE OctToStr (VAR S : AnsiString;Value,Digits : CARDINAL);`

Description: Convert `Value` to an octal representation and write it to `S`, the procedure always writes `Digits` digits. If you specify more `Digits`, the result will be left padded with zeroes. If you specify `Digits` less than the number of bits needed to represent `Value`, only the rightmost `Bits` digits will be written. (like `Value` was ANDed with $2\hat{3}$*`Digits`-1)

The `Digits` parameter can lie in the range 0..32, but 0 does nothing.

Errors: Pascal string : None, AnsiString: Untested

See also: StrToBinary (64) StrToOct (64) StrToHex (65) OctToStr (65) HexToStr (66)

For an example, see BinaryToStr (65)

9.1.23 HexToStr

Declaration: `PROCEDURE HexToStr (VAR S : String;Value,Digits :  CARDINAL);`
`PROCEDURE HexToStr (VAR S : AnsiString;Value,Digits :  CARDINAL);`

Description: Convert `Value` to an hexadecimal representation and write it to `S`, the procedure always writes `Digits` digits. If you specify more `Digits`, the result will be left padded with zeroes. If you specify `Digits` less than the number of bits needed to represent `Value`, only the rightmost `Bits` digits will be written. (like `Value` was ANDed with $2\hat{(}4$*`Digits`)-1)

The `Digits` parameter can lie in the range 0..32, but 0 does nothing.

Errors: Pascal string : None, AnsiString: Untested

See also: StrToBinary (64) StrToOct (64) StrToHex (65) OctToStr (65) HexToStr (66)

For an example, see BinaryToStr (65)

9.1.24 LGrow

Declaration: `PROCEDURE LGrow(VAR S: STRING;Ch:CHAR;Count:LONGINT);`
`PROCEDURE LGrow(VAR S: AnsiString;Ch:CHAR;Count:LONGINT);`

Description: If `Length` (() S)¡`Count` then pad at the beginning of the string with character `C` until `Length(S)` (=) `Count`

Errors: Pascal string : None, AnsiString: Untested

See also: RGrow (67)

```
Uses EPasStr;

VAR P : String;

BEGIN
  P:='1';
  LGrow(P,' ',10);
  Writeln(P);            { Writes '         1'}
  RGrow(P,' ',20);
  Writeln(P);            { Writes '         1          '}
END.
```

9.1.25 RGrow

Declaration: `PROCEDURE RGrow(VAR S : String;C:CHAR;Count:LONGINT);`
`PROCEDURE RGrow(VAR S : AnsiString;C:CHAR;Count:LONGINT);`

Description: If `Length` (() S)¡`Count` then pad at the end of the string with character `C` until `Length(S)` (=) `Count`

Errors: Pascal string : None, AnsiString: Untested

See also: LGrow (66)

For an example, see LGrow (66)

9.1.26 StrStr

Declaration: `PROCEDURE StrStr(VAR P:String;C:Char;Count:LONGINT);`
`PROCEDURE StrStr(VAR P:AnsiString;C:Char;Count:LONGINT);`

Description: Fill `P` with `Count` times `C`. (the old basic function String$)

Errors: Pascal string : None, AnsiString: Untested

```
Uses EPasStr;

VAR P : String;

BEGIN
  P:='1';
  StrStr(P,' ',10);
  Writeln(P);          { Writes '          '}
END.
```

9.1.27 Item procedures

Declaration: `PROCEDURE Item(VAR R: STRING; CONST S: STRING; T: CHAR; N: LONGINT);`
`PROCEDURE Item(VAR R: STRING; CONST S: STRING; CONST T: CHARSET; N: LONGINT);`
`PROCEDURE ItemS(VAR R: STRING; CONST S: STRING; CONST T: String; N: LONGINT);`
`PROCEDURE Item(VAR R: AnsiString; CONST S: AnsiString; T: CHAR; N: LONGINT);`
`PROCEDURE Item(VAR R: AnsiString; CONST S: AnsiString; CONST T: CHARSET;`
`N: LONGINT);`
`PROCEDURE ItemS(VAR R: AnsiString; CONST S: AnsiString; CONST T: String;`
`N: LONGINT);`

Description: These procedures are variations on the same theme. The procedure parses `S`, and returns the `N`th substring which is delimited with characters from `T` in R, and empties R if no such such substring exists. The variable `T` contains the separator characters, which can be one single character, a CHARSET (=SET OF CHAR) or a string with characters(ItemS).

The ItemS procedure merely converts the string to a CHARSET and then calls the CHARSET procedure.

I use these procedures a lot. Specially when processing textfiles.

Errors: Pascal string : None, AnsiString: Untested

See also: GetBetween (68)

```
Uses EPasStr;

VAR Source, Dest : String;
    A :WORD;

BEGIN
 Source:=' hello1  hello2  hello3  hello4 ';
 FOR A :=0 TO 4 DO
  BEGIN
   Write(A,' ');
   Item(Dest,Source,' ',A);
   IF Length(Dest)=0 THEN
    Writeln('Empty')
   ELSE
    Writeln(Dest);
  END;
END.
{
Prints:
 0 hello1
 1 hello2
 2 hello3
 3 hello4
 4 Empty
 }
```

9.1.28 GetBetween

Declaration: `FUNCTION GetBetween(Source:String;VAR Dest:String;C1,C2:CHAR):BOOLEAN;`
`FUNCTION GetBetween(Source:AnsiString;VAR Dest:AnsiString;C1,C2:CHAR):BOOLEAN;`

Description: Returns in `Dest` the string between the first occurance of C1 in `Source` and the first occurance of C2 AFTER the 1st occurance of C1. If C1=C2, the procedure returns the string between first and second occurance of C1.

Returns status (existance of both characters and position C1¡ C2)

Errors: Pascal string : None, AnsiString: Untested

See also: Item procedures (67)

```
Uses EPasStr;

VAR Source, Dest : String;

BEGIN
 Source:='0123456';
 GetBetween(Source, Dest, '1', '4');
 Writeln(Dest);                   { Writes '23'}
END.
```

9.1.29 CommaStr

Declaration: `PROCEDURE Commastr(var S : String;sep:CHAR);`
`PROCEDURE Commastr(var S : AnsiString;sep:CHAR);`

Description: Inserts `sep` on every 3rd spot relative to the end of string `S`. (e.g. S:='123456789'; CommaStr(S,',') -¿ S:='123,456,789')

Errors: Assembler version not functional or buggy?

```
Uses EPasStr;

VAR Source, Dest : String;

BEGIN
  Source:='0123456';
  GetBetween(Source, Dest, '1', '4');
  Writeln(Dest);                    { Writes '23'}
END.
```

9.1.30 CompressTabs

Declaration: `PROCEDURE Compresstabs(CONST Source:String;VAR Dest:String;Tabsize:LONGINT);`
`PROCEDURE Compresstabs(CONST Source:AnsiString;VAR Dest:AnsiString;Tabsize:LONGINT);`

Description: Compress tabs to spaces, with variable tabsize. This procedure doesn't simply compress `Tabsize` spaces to a hardtab, but implements tabbing like in an ordinary texteditor like q.exe (or joe).

Doesn't function well with hardtabs already in the input-string Source. In that case, run ExpandTabs (69) first.

Source string is untouched. 'Normal' `Tabsize` is 8, but specially in programmers editors, it is often 2 or 4.

Errors: Assembler version not functional or buggy?

See also: ExpandTabs (69)

```
Uses EPasStr;

VAR P : String;

BEGIN
  P:='12345678';
  CommaStr(P, '.');
  Writeln(P);            {writes '12.345.678'}
END.
```

9.1.31 ExpandTabs

Declaration: `PROCEDURE ExpandTabs(CONST P : String;VAR P2:String;Tabsize:LONGINT);`
`PROCEDURE ExpandTabs(CONST P : AnsiString;VAR P2:AnsiString;Tabsize:LONGINT);`

Description: Expands tabs in `P` to spaces, puts result in `P2`. (`P` untouched). `Tabsize` is the number characters between two tabs.

This procedure implements real detabbing, not simply replacing a hardtab with `tabsize` spaces. It doesn't implement smart tabbing(place tabstops dependant on text on a previous line).

This procedure is used to deal with tabs when reading textfiles. One ExpandTabs after each stringread (ReadLn) from the file, and forget all problems with tabs. If the identation doesn't matter, you can just use ReplaceChar (60) (which would be faster) and replace each tab with a space.

Errors: Assembler version not functional or buggy?

See also: CompressTabs (69) ReplaceChar (60)

For an example see CompressTabs (69)

9.1.32 Invert

Declaration: `PROCEDURE Invert( VAR s :String);`
`PROCEDURE Invert( VAR s :AnsiString);`

Description: Inverts the string in `s`. ('ABCD' -¿ 'DCBA'). I don't know what it's good for, except maybe palindrome checking, but I had it lying around somewhere.

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='12345678';
 Invert(P);
 Writeln(P);            {writes '87654321'}
END.
```

9.1.33 RPos

Declaration: `FUNCTION RPos(CONST SubStr,S : String):LONGINT;`
`FUNCTION RPos(CONST SubStr,S : AnsiString):LONGINT;`

Description: Inverts the string in `s`. ('ABCD' -¿ 'DCBA'). I don't know what it's good for, except maybe palindrome checking, but I had it lying around somewhere. When `SubStr` contains only one char RCharPos (61) is called, which is faster.

Errors: None.

See also: RCharPos (61) NextRCharPos (62)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='This is a test. A test for RPos I mean.';
 Writeln('Pos  : ',Pos('test',P));   {writes 11 }
 Writeln('RPos : ',RPos('test',P));  {writes 19 }
END.
```

9.1.34 ReplaceLast

Declaration: `FUNCTION ReplaceLast(CONST Sub1,Sub2: STRING;SS: STRING ): STRING;`
`FUNCTION ReplaceLast(CONST Sub1,Sub2: AnsiString;SS: AnsiString): AnsiString;`

Description: Replaces the last occurance of `Sub1` in `SS` with `Sub2` and returns the result. (The procedures searches for `Sub1` starting from the end of `SS`)

Errors: None.

See also: Replace (71) ReplaceChar (60)

```
Uses EPasStr;

VAR P : String;

BEGIN
 P:='Hello hello _hello_';
 p:=ReplaceLast('hello','HELLO',P);
 Writeln(P);
END.
```

9.1.35 Replace

Declaration: `FUNCTION Replace(CONST Sub1,Sub2: STRING;SS: STRING ): STRING;`
`FUNCTION Replace(CONST Sub1,Sub2: AnsiString;SS: AnsiString): AnsiString;`

Description: Replaces the first occurance of `Sub1` in `SS` with `Sub2` and returns the result. (The procedures searches for `Sub1` starting from the start of `SS`)

Errors: None.

See also: ReplaceLast (71) ReplaceChar (60)

For an example see ReplaceLast (71)

9.1.36 LeftStr

Declaration: `FUNCTION LeftStr( CONST StrName: String; NumChars : Integer) : String;`
`FUNCTION LeftStr( CONST StrName: AnsiString; NumChars : LONGINT) : AnsiString;`

Description: Old basic procedure Left$, provided for easy porting, some people prefer this syntax above Copy (??). Copies the first `NumChars` characters from `StrName`. (Rougly equivalent Result=Copy(S,1,NumChars))

Errors: None.

See also: MidStr (72) RightStr (72) Slice (72)

```
Uses EPasStr;

VAR P,P2 : String;

BEGIN
```

```
 P:='This is a test. A test for RPos I mean.';
 Writeln('First 9 chars on the left         : ',LeftStr(P,9));
 Writeln('First 9 chars on the right        : ',RightStr(P,9));
 Writeln('4 characters on position 11       : ',MidStr(P,11,4));
 Slice(P2,P,11,4);                   {Faster than MidStr,
                                      no stringcopy except the needed 4 chars}
 Writeln('Same with Slice (procedure form)  : ',P2);
END.
```

9.1.37 RightStr

Declaration: `FUNCTION RightStr( CONST StrName: String; NumChars : Integer) : String;`
`FUNCTION RightStr( CONST StrName: AnsiString; NumChars : LONGINT) : AnsiString;`

Description: Old basic procedure Right$, provided for easy porting, some people prefer this syntax above Copy (??). Copies the last `NumChars` characters from `StrName`. (Rougly equivalent Result=Copy(S,Length(S)+1-NumChars,NumChars))

Errors: None.

See also: MidStr (72) LeftStr (71) Slice (72)

For an example see LeftStr (71)

9.1.38 MidStr

Declaration: `FUNCTION MidStr( CONST StrName: String; StartPos, NumChars : LONGINT) : String;`
`FUNCTION MidStr( CONST StrName: AnsiString; StartPos, NumChars : LONGINT) : AnsiString;`

Description: Old basic procedure Mid$, provided for easy porting, some people prefer this syntax above Copy (??). Returns the first `NumChars` characters starting on `StartPos`. (exactly the same as Copy)

Errors: None.

See also: RightStr (72) LeftStr (71) Slice (72)

For an example see LeftStr (71)

9.1.39 Slice

Declaration: `PROCEDURE Slice (VAR R: String; CONST S: String; P,L: LONGINT);`
`PROCEDURE Slice (VAR R: AnsiString; CONST S: AnsiString; P,L: LONGINT);`

Description: Modula-2 version of Copy (??). Equivalent to R:= Copy (??)(S,P,L); Also faster than Slice, since it has a procedure-form instead of a function. (A function requires an extra copy of the string to

Errors: None.

See also: RightStr (72) LeftStr (71) MidStr (72)

For an example see LeftStr (71)

9.1.40 Match

Declaration: `FUNCTION Match(CONST source, pattern: String): Boolean;`
`FUNCTION Match(CONST source, pattern: AnsiString): Boolean;`

Description: String matching procedure (capable of * and ?, so things like ncurses*.source.*.rpm matches ncurses-3.0.source.i386.rpm) Speed seems satisfactory despite recursion.

Errors: Reads sometimes byte at source[length(source)+1], same for pattern. Ansistring version untested.

```
Uses EPasStr;

PROCEDURE Quicktest(S1,S2:String);

BEGIN
 Write(S1,'   :   ',S2,'          ');
 IF Match(S2,S1) THEN
  Write('ok')
 ELSE
  Write('Failure');
 Writeln;
END;

BEGIN
 Quicktest('*ll','1stpat.testfile.ll');
 Quicktest('*testf*.ll','1stpat.testfile.ll');
 Quicktest('*txestf*.ll','1stpat.testfile.ll');
 Quicktest('*.test*.ll','1stpat.testfile.ll');
END.
```

Chapter 10

The EWindow unit.

This chapter describes the `EWindow` unit, which works with both FPC Pascal (Go32V2 and Linux tested) and Borland Pascal 7.0 (you'll need the libsrc/bpgo32.pas unit). It will probably work on TP 6.0 too, but this hasn't been tested.

10.1 unit EWindow

One of the most interesting standard units of the Modula2 RTL (also a Wirth language like Pascal) is module Window. It implements multithreaded textwindowing. So this module does NOT exist in the modula2 version of XTDLIB, it was written from scratch while peeking at the several Modula2 implementations.

10.1.1 Additional remarks, bugs and principles

This module emulates the Modula2 counterpart A BIT. The module is still under development, but the raw core stands, the rest is changing window colors, implementing shades etc, moving etc.

Opposed to the Modula2(TopSpeeds, not the ISO) version, this module is

- NOT multithreading
- EWindow doesn't handle it's own Crt like M2-Window, but is build on top of FPC-Pascal's Crt for normal operations, and uses direct-screenwrites to update the screen after major (window opening/closing/moving/hiding) changes.

  The Linux version uses GotoXY, TextAttr and Textcolor and Write to update the screen, which is a lot slower, but not a problem with computer speeds equal or above 66 MHz.

  As a result of this

  – It's impossible to implement wordwrapping via normal procedure writeln (yet) However a separate procedure WrapWrite (85) implements wrapping in a window.
  – Writing to hidden or not activated windows is not possible. The only window that can be written is the one on top. Each window (including the background, the (FullScreen (80)-window) can be put on top however. A separate routine could be made for writing to hidden/obscured

windows, but I haven't thought really well about how to implement that yet.

Other comments:

- Unportable as it is, unless your OS supports some kind of virtual screen like Dos'. I hope to create a Linux version based on Ncurses sometime, but it will be almost totally different internally. In the mean time, I implemented a Linux version without Ncurses, which turned out better (read faster) than I expected. It's definitely faster than textwindow environments as seen when you use make menuconfig, since my unit doesn't redraw the entire screen after each write (an huge part of that credit goes to the Linux Crt implementor btw, Great job!).

  I'm currently asking questions about OS/2 support, and it seems to be possible, even easy, however installing OS/2 (to create a testenvironment) is a bit problematic.

- A lot of the non-basic functions (frame-color changes, frameon/off, snapshots without opening a window etc) aren't implemented yet. Some basic checks are now implemented, but the errorhandler is simply an halt Error Handling (78)

  The basic routines(Open,Use,Close,Clone,Hide,Unhide) work however.

- Screen redrawal has greatly improved. The algoritm isn't perfect yet see see Ewindows Internal (77) but the unoptimalities are small. I'm thinking about a small optimalisation to reduce flickering of the screen with a high (¿50) number of windows under dos.

Advantages compared to Modula2 version, and to an average equivalent from SWAG (the latter marked with `PAS`)

- Commandline use (popup window, do your thing in that window, popoff window, screen seems untouched) is still possible contrary to the M2-version of EWindow which cleared the screen on start-up.

  `not for linux`

- `PAS` FullScreen (80) is just another window, fully compatible with any other window (the same operations can be used), and you can have more than one.(The default FullScreen is no different then a window you open, except that under Dos the contents of the screen when starting the program are copied to this particular window)

  If you switch to FullScreen (Use(FullScreen)), or another big (Width*Height) window before a dosshell (e.g. ARJ), you can capture the last 25 lines of output, without redirection. The screen below all other screens is empty black by default.

  I'm not sure if this works under Linux, but almost everything is pipable under Linux, so this isn't a big problem

- Little size dependant. If the detection (screensegment, Width and height) is ok, everything should work, the actual code is compatible.

- `PAS` "Unlimited" windows, no arrays with data. I let Windtest open 500 windows, no problem at all, except for some flickering when you run without delays. (You can test that yourself, simply increase the RanWinNr constant in windtest.

- `PAS` Full Crt compability. GotoXY in a Window, TextColor in a window, Cursors and attributes are saved when switching/moving windows. Overlapping windows are no problem. (Linux version doesn't save cursorstate)
- `PAS` Handle based (sometimes called "OOP without OOP"). Mainmodule has to regard virtually no details. The unit takes care of almost everything, the mainmodule only supplies the parameters, and the module does ALL the work, see relative simplicity of WindTest.
- `PAS` Much safer, because all details are hidden. In M2 I let user move windows on the screen to the place where THEY want them. Simply with Scroll-lock and cursorkeys. This requires about 20 lines code in the mainmodule.
- `PAS` Rather OS-independant, dependancies in 1 1/2 procedure, and a few lines of less important conditional code (cursorstates, screendimension detection) in other procedures. Also the strict interface-implementation separation makes applications using this unit a bit OS-independant, if you have EWindow for a lot of OS'es

10.1.2 Project status

- The old Runtime-204 bug has been fixed, there are no major bugs as far as I know.
- A very simple errorhandling system has been setup to detect corrupted windows and weird coordinates

  The system isn't waterproof, and can be turned of with a conditional. If you however use good coordinates, everything works. This system is temporarily I think. If you have suggestions how to handle this kind of errors (create runtime error, do nothing on error(but try to fix data), maybe even signals), let me know.
- `FPC DOS/Go32v2` When you use crazy numbers of windows(¿50,100), and open and close them without doing much (which delays), the screen can flicker a bit. I tried to reduce this by buffering output on the stack, but that wasn't so successful. Anyway, the routine which caused the flicker doesn't reflect a real application anyway, it is more a benchmarking/testing routine.

  Dos version opens and closes 500 windows in 5 seconds on a Cyrix P166+
- `FPC DOS/Go32v1` Status unknown. I use RTL procedures Go32.DosMemGet and -Put, but these do exist under Go32V1. Another incompability might be the assembler procedures (which set/get the cursorsize) for Dos in ELib. If you test under Go32V1, let me know.
- `Borland Pascal DOS` (no bugs, except that FPC-Crt standard issues highvideo, and Bp-Crt not, which results in blinking attributes.)

  I don't test BP as much as the other OSes, so maybe the newest additions may cause some problems.
- `FPC Linux` Writing to bottom right character (coordinates Width,Height) scrolls the screen and messes up screen. Haven't found a working workaround yet.

  Linux version seems fast enough now, but I'm running on a Cyrix P166+

  The unit is more than usable under Linux, and even faster than make menuconfig (textwindowed kernel-configuration, see your kernel source). Make

Menuconfig updates the screen to often. The Crt of the FPC/Linux is very good, and since Ewindow is based on it, it inherites those advantages.

I haven't tried running a program that uses EWindow via a telnet connection, since my second computer is offline.

The release 0.99.8 requires Crt.pp to be patched (See devel directory, only change: virtual screen moved from implementation to interface), newer snapshots (0.99.9 since mid-october) won't have this problem.

- Other OS'es not implemented yet, I just wrote a letter to Dan for OS/2 help, and got some response. At least now I'm sure it's possible, and not extremely much work. The OS/2 port will depend on me be able to test under OS/2. IOW, do I get OS/2 installed? :-)

10.1.3 EWindow internal

I gave the idea of speeding up module EWindow a bit thought, looked at the TopSpeed RTL, and wondered how they solved it. The trick seems to be to build the windowlist (a linked list of the records with window-information, of type WinType (79) from top to bottom instead of bottom to top, but I can't figure out exactly how they do it (pretty complicated code, multithreading and writes to hidden windows)

My `old approach` when redrawing a screen is to

- Clear the screen, just in case the windows don't cover the entire screen
- Write the "lowest" window.
- Write the but lowest window
- etc etc until the top window

The above method is optimized by buffering the data to be written on the stack (instead of directly writing to the videomemory), and by only rewriting the rectangle that needs to be updated.

The new approach (since 0.10 or so. On a per line basis, and only the part that has to be redrawn, redrawscreen(1,1,Width,Depth) rewrites the entire screen. This is the general idea, some details (like which tests should be greater AND equal instead of only greater) aren't described here) :

- Temporary values, TX1:=X1; TX2:=X2; where X1-X2 is the range of the current (partial) line we are rewriting
- Get a (first : top) window. 5 Cases :
  - window.X2 smaller than TX1. Do nothing, that window not in current range
  - window.X1 bigger than TX2. Do nothing, that window not in current range
  - Window.X1 smaller than TX1 TX1 < Window.X2 < TX2 Rewrite TX1..Window.X2 from buffer then Tx1:=Window.X2 (since original Tx1 ..Window.X2 is now in it's final state)
  - Window.X2 bigger than TX2 TX1 > Window.X1 > TX2 Rewrite Window.X1..TX2 from buffer then Tx2:=Window.X1

  - Both Window.X1 and Window.X2 lie in the TX1 and TX2 interval. Don't know how to solve this yet. Redraw Window.X1..Window.X2 and then recursively redraw X1..Window.X1 and Window.X2..X2? I can't figure out how TopSpeed solves it, but I don't see recursion.

- IF TX1 <> TX2 and Window.Next <> NIL take nextwindow and repeat step 2

(Added later) I see know how the TopSpeed guys do it. It's not recursive, but not really with less overhead. They kind of use a bi-section system. But I'm not sure what TopSpeeds system will do in this case:

```
|      yyyyyyyy         xxxxxxxxx   zzzzzzzzz   |
```

(legenda:)

- the region between both pipesigns is to be rewritten.
- the x window is more on top then y, and is written first.
- after writing the x window the routine saves the position of the most right x in a variable NextX, and the distance NextX to last pipesign in NextLen
- Then in reprograms the coordinates to do the area left of x.
- y is written, distance right y and distance right-y - left x is stored in NextX and NextLen, and THAT's my problem. At that moment it wipes out the variables which have to govern writing the region right of X.

This is a rare case, and the TopSpeed programmers only redraw parts of windows at once, and check for a lot of operations when redrawing (they don't redraw the screen using the same procedure). Like Hide. Hide uses a specialised procedure which draws anything below the window to be hidden.

On the other hand, the TopSpeed RTL is(was) to good to allow such bugs. I used this module for years, and never saw one flaw. What am I missing! At least I see why such complicated textwindowing units aren't included in SWAG :-)

I still will try to implement above algoritm, since it's the best, but for inbetween I implemented another system, which is much faster than the old one, but a bit slower than the new algoritm, but simpler, mainly because it doesn't require the windowlist (and with that a lot procedures more) to be changed.

The `current` implementation also only redraw the parts of the screen that have changed, but writes bottom up, which is slower because all windows in the range which is rewritten, are redrawn, instead of only the top ones. Under Dos this is no problem. Under Linux, with it's terminal driven screen it is, so I changed the unit to do the building up of the screen in (fast) memory, not directly on the screen, which is a lot faster.

Maybe it's slow on 386/33's etc, but above 486 DX2/66 speed won't be a problem.

10.1.4 Error handling

If the conditional WindowCheck is defined, some errorhandling is implemented, by using Thomas Schatzl ErrHndler unit. (the same he uses for DPMI)

The unit checks for some errorconditions like corrupted WinTypes, bad coordinates (X1 bigger than X2), and reports error to the errorhandler.

The default errorhandler cleans the screen, turns of Window()'ed mode and puts an oneliner error message on the screen.

10.2 Types

10.2.1 Styles

Declaration: `(Style definitions are not shown because it contains a lot of high ascii, and will get mangled)`

Description: Styles are small strings with all characters needed to make frames in textwindows. The exact definition may change, so please use one of the predefined styles if you can.

See also: WinDef (79)

10.2.2 Coordinates

Declaration: `AbsCoord= INTEGER; RelCoord= INTEGER;`

Description: As you can see the above definitions aren't very complex, their main purpose is to let clearly see in the definition what are coordinates for the entire screen (e.g. window-positions), and which ones are positions IN windows. (e.g. cursorposition).

See also: WinMove (83), Change (82) WinDef (79)

10.2.3 WinType

Declaration: `WinType= ŴinDataRec`

Description: THE main type of this unit. A WinType is a type which describes a certain window, and is used to reference it. You should never directly manipulate a WinType parameter, or the record it points to (WinDataRec). Use EWindow procedures to achieve what you want.

See also: WinOpen (80), WinClose (81), FullScreen (80)

10.2.4 WinDef

Declaration:
```
WinDef      = RECORD
               X1,Y1,
               X2,Y2        : AbsCoord;   Coordinates of Window to create
               Foreground   : BYTE;       Initial colors of window
               Background   : BYTE;
               FrameOn      : BOOLEAN;    Should a frame be created for the window?
               FrameDef     : PChar;      Bordertype, POINTER TO a \seetypl{Style}{Styles}
               FrameFore    : BYTE;       Colors of the frame
               FrameBack    : BYTE;
               END;
```

Description: A WinDef record contains all data needed to open a new window.

See also: WinOpen (80)

10.3 Variables

10.3.1 Height and Width

Declaration: `VAR Width,`
`Height : INTEGER;`

Description: Height and Width are the dimensions of the textmode screen. Under the Go32V2 memorymodel the initial values are copied from BIOSData area, and under Linux from (new) Linux Crt's variables ScreenHeight and ScreenWidth.

New, because the old Linux Crt (before nov 1998) had no support for other dimensions.

See also: None, but used by most procedures that have coordinates as parameter.

10.3.2 FullScreen

Declaration: `VAR FullScreen :` WinType (79)`;`

Description: When EWindow starts up, the entire textscreen is saved (Go32V2 only) to the (frameless) FullScreen window. Under Linux the FullScreen inits as empty.

The Fullscreen is an ordinary window, just like ones you open yourself with WinOpen (80). You can hide it, clone it etc. You can use a fullscreen to switch to (Use (81)) before an dos-shell and capture the last 25 lines of output.

Most times you use it as background, or to restore the dos screen after your application is finished, or clone (84) it and use it for both:-)

10.4 Functions and procedures

10.4.1 WinOpen

Declaration: `FUNCTION WinOpen(WD :` WinDef (79)`):` WinType (79)`;`

Description: Opens a window described by a WinDef (79), and puts it on top.

The returnvalue is used by the unit for further operations on that window. You'll have to save it, and pass it to almost every other EWindow procedure.

Note: FullScreen (80) window is already WinOpen'ed at startup

Errors: None.

See also: WinClose (81), WinClone (84), Use (81)

```
Uses EWindow, Crt;

CONST WD: WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
                  Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);


VAR Win : WinType;

BEGIN
 Win:=EWindow.WinOpen(WD);        {Open the window}
```

```
  SetTitle(Win,'Window 1');          {Set the title on this framewindow (FRAMEON=TRUE)}
  Readln;
  EWindow.WinClose(Win);             {Close the @££%@ Window}
END.
```

10.4.2 WinClose

Declaration: `PROCEDURE WinClose(VAR W: WinType (79));`

Description: Closes the Window associated with the WinType. NIL is assigned to the WinType parameter.

Note: You can WinClose the FullScreen (80) window if you wish, but space on the screen not covered by a window will be empty black.

Errors: None.

See also: WinOpen (80), Hide (82)

See WinOpen for an example.

10.4.3 SetTitle

Declaration: `PROCEDURE SetTitle(W: WinType (79);Title:PChar);`

Description: Sets the title of the window, only use this on Frame-windows.(otherwise it would do nothing) The title is showed in the upper frame-line (centered).

Errors: None.

See also: Oops. No really related procedure I think.

See WinOpen (80) for an example.

10.4.4 Use

Declaration: `PROCEDURE Use(VAR W: WinType (79));`

Description: Puts the window on top, so that it can be written. Window must be WinOpen (80) first. The USEd window is the window which will be written by standard output (Write(Ln)).

Errors: None.

See also: WinOpen (80), WinClone (84), UnHide (82)

```
Uses EWindow,Crt;

CONST WD:  WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
                   Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);

CONST WD2:  WinDef=(X1:12;Y1:15;X2:48;Y2:23;Foreground:yellow;Background:white;
                   Frameon:TRUE; FrameDef:Style2;FrameFore:Yellow;FrameBack:Blue);


VAR Win,Win2 : WinType;
```

```
BEGIN
  Win:=EWindow.WinOpen(WD);          {Open the window}
  Win2:=Ewindow.WinOpen(WD2);      {opens a second window, which is on top}
  Use(Win);                          {Puts window 1 back on top, so it can be written}
  EWindow.WinClose(Win2);            {You don't have to use a window to close it}
  EWindow.WinClose(Win);             {Close the other Window}
END.
```

10.4.5 Hide

Declaration: `PROCEDURE Hide(W: WinType (79));`

Description: Hides the window (makes it invisible) Makes the next window in line on top, but that is not 100after a Hide command.

Errors: None.

See also: WinOpen (80), WinClone (84), UnHide (82) WinClose (81)

```
Uses EWindow,Crt;

CONST WD: WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
                  Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);


VAR Win : WinType;
BEGIN
  Win:=EWindow.WinOpen(WD);        {Open the window}
  Hide(Win);                         {makes Window invisible}
  ReadLn;
  UnHide(Win);                          {Makes Window visible again}
  Readln;
  EWindow.WinClose(Win);             {Close the @££%@ Window}
END.
```

10.4.6 UnHide

Declaration: `PROCEDURE UnHide(W: WinType (79));`

Description: Unhides the window (makes it visible again). The procedure doesn't put the window on top. You should issue a Use (81) if you want to write to the screen.

Errors: None.

See also: WinOpen (80), WinClone (84), Hide (82) WinClose (81)

See Hide for an example.

10.4.7 Change

Declaration: `PROCEDURE Change(W: WinType (79); OX1,OY1,OX2,OY2: AbsCoord (79));`

Description: Change changes the dimensions and/or position of window `W` to top left corner (OX1,OY1) and bottom right corner (OX2,OY2).

Only contents which fit in new window are copied, if the new window is bigger, the rest is filled with spaces with the current attributes for that window. If you resize to a smaller dimension, the surplus is lost

Errors: None.

See also: WinMove (83)

```
Uses EWindow, Crt;

CONST WD: WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
                  Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);

VAR Win : WinType;

BEGIN
 Win:=EWindow.WinOpen(WD);          {Open the window}
 ReadLn;
 Change(Win,1,10,20,20);              {Change dimensions to (1,10) (20,20)}
 ReadLn;
 EWindow.WinClose(Win);             {Close the @££%@ Window}
END.
```

10.4.8 WinMove

Declaration: `PROCEDURE WinMove(W: WinType (79) ; OX1,OY1: AbsCoord (79));<p>`

Description: WinMove move a window so that top left of the window is on position (OX1,OY1)¡p¿ WinMove is faster than Change (82) cause it doesn't resizes.

Errors: None.

See also: Change (82)

```
Uses EWindow, Crt;

CONST WD: WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
                  Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);

VAR Win : WinType;

BEGIN
 Win:=EWindow.WinOpen(WD);          {Open the window}
 ReadLn;
 WinMove(Win,1,10);              {Change dimensions to (1,10) (20,20)}
 ReadLn;
 EWindow.WinClose(Win);             {Close the @££%@ Window}
END.
```

10.4.9 Clear

Declaration: `PROCEDURE Clear(W: WinType (79));`

Description: Clears a window with the proper attributes, and performs a GotoXY(1,1);
Crt.ClrScr may seem work right in Dosmode, but maybe it won't on other platforms.
Clear is also faster, and allows you to clear a window which is not on top.

Errors: None.

See also: Can't think of a reasonable link here.

```
Uses EWindow, Crt;

CONST WD: WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
                  Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);

VAR Win : WinType;

BEGIN
 Win:=EWindow.WinOpen(WD);          {Open the window}
 Writeln('text');
 ReadLn;
 Clear(Win);                        {Clear the window}
 ReadLn;
 EWindow.WinClose(Win);             {Close the @££%@ Window}
END.
```

10.4.10 WinClone

Declaration: `FUNCTION WinClone(W: WinType (79)): WinType (79);`

Description: Clone window W, (creates a full copy, doesn't just return a pointer to the same window) hides de clone and return a WinType to the cloned window.

Clones can simply get closed with WinClose (81)

Errors: None.

See also: WinOpen (80) WinClose (81)

```
Uses EWindow, Crt;

CONST WD: WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
                  Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);

VAR Win, Win2 : WinType;

BEGIN
 Win:=EWindow.WinOpen(WD);          {Open the window}
 ReadLn;
 Win2:=WinClone(Win);               {Clear the window}
 Writeln('text');                   {Write text to the original window}
 WinMove(Win2,10,10);               {Move Win2 to another position}
 Unhide(Win2);                      {Unhide it}
 Use(Win2);                         {Put on top and redirect output to it}
 writeln('Not the same text');
 ReadLn;
 EWindow.WinClose(Win);             {Close the @££%@ Window}
 EWindow.WinClose(Win2);            {Close the @££%@ Window}
END.
```

10.4.11 WrapWrite

Declaration: `PROCEDURE WrapWrite(CONST Towrite:String);`
`PROCEDURE WrapWriteLn(CONST Towrite:String);`

Description: Writes string `Towrite` to the current Window, and tries to wrap the text in it. WrapWriteLn adds a linefeed.

Errors: None.

See also: none.

```
Uses EWindow, Crt;

CONST WD: WinDef=(X1:10;Y1:13;X2:45;Y2:25;Foreground:White;Background:Black;
              Frameon:TRUE; FrameDef:Style1;FrameFore:Yellow;FrameBack:Blue);


VAR Win : WinType;

BEGIN
 Win:=EWindow.WinOpen(WD);        {Open the window}
 WrapWriteln('A longer text than which will fit into this window on one line I hope'
 ReadLn;
 EWindow.WinClose(Win);           {Close the @££%@ Window}
END.
```

Chapter 11

Farmem Unit

This is the documentation of the Farmem unit.

The farmem unit is a small unit which provides access to other memory segments (then your own programs) by defining a series classes with properties which resemble an array.

Two basic classes exist,

- `tdosmem` to access the dosmemory, which is a special case, because under FPC (Go32V2) %fs always points to the dos-segment.
- `tfarmem` to access a custom segment.

Both basic types come each in three flavours, a array of byte, array of word and array of longint approach, conveniently denoted with suffix b,w and l.

11.1 Class defintions

11.1.1 tdosmemb

Declaration:

```
 tdosmemb = class
        procedure writemem(ofs : DWord; data : byte);
        function readmem(ofs : DWord) : byte;
        property memarray[ofs : DWord] : byte read readmem write writemem; default;
end;
```

Description: Basic class to access a different (array of byte typed) segment via register %fs, which points to realmode dos memory under FPC.

See also: tdosmemw (86) tdosmeml (87) tfarmemb (87) tfarmemw (87) tfarmeml (88)

11.1.2 tdosmemw

Declaration:

```
 tdosmemw = class
        procedure writemem(ofs : DWord; data : Word);
        function readmem(ofs : DWord) : Word;
        property memarray[ofs : DWord] : Word read readmem write writemem; default;
end;
```

Description: Basic class to access a different (array of word typed) segment via register %fs, which points to realmode dos memory under FPC.

See also: tdosmemb (86) tdosmeml (87) tfarmemb (87) tfarmemw (87) tfarmeml (88)

11.1.3 tdosmeml

Declaration:

```
  tdosmeml = class
                procedure writemem(ofs : DWord; data : Longint);
                function readmem(ofs : DWord) : Longint;
                property memarray[ofs : DWord] : Longint read readmem write writemem; default;
 end;
```

Description: Basic class to access a different (array of longint typed) segment via register %fs, which points to realmode dos memory under FPC.

See also: tdosmemb (86) tdosmemw (86) tfarmemb (87) tfarmemw (87) tfarmeml (88)

11.1.4 tfarmemb

Declaration:

```
  tfarmemb = class
                procedure writemem(s : Word; ofs : DWord; data : byte);
                function readmem(s : Word; ofs : DWord) : byte;
                property memarray[s : Word; ofs : DWord] : byte read readmem write writemem; defau
 end;
```

Description: Basic class to access a different (array of byte typed) segment via seg s.

The segment is loaded to register %gs, and register

The segment probably wasn't implemented as a variable (in the class definition) because FarMem is implemented in assembler, and FPC's calling conventions of objects have changed a lot lately

See also: tdosmemb (86) tdosmemw (86) tdosmeml (87) tfarmemw (87) tfarmeml (88)

11.1.5 tfarmemw

Declaration:

```
  tfarmemw = class
                procedure writemem(s : Word; ofs : DWord; data : word);
                function readmem(s : Word; ofs : DWord) : word;
                property memarray[s : Word; ofs : DWord] : word read readmem write writemem; defau
 end;
```

Description: Basic class to access a different (array of word typed) segment via seg s.

The segment is loaded to register %gs, and register

The segment probably wasn't implemented as a variable (in the class definition) because FarMem is implemented in assembler, and FPC's calling conventions of objects have changed a lot lately

See also: tdosmemb (86) tdosmemw (86) tdosmeml (87) tfarmemb (87) tfarmeml (88)

11.1.6 tfarmeml

Declaration:
```
 tfarmeml = class
         procedure writemem(s : Word; ofs : DWord; data : longint);
         function readmem(s : Word; ofs : DWord) : longint;
         property memarray[s : Word; ofs : DWord] : longint read readmem write writemem; de
end;
```

Description: Basic class to access a different (array of longint typed) segment via seg `s`.

The segment is loaded to register %gs, and register

The segment probably wasn't implemented as a variable (in the class definition) because FarMem is implemented in assembler, and FPC's calling conventions of objects have changed a lot lately

See also: tdosmemb (86) tdosmemw (86) tdosmeml (87) tfarmemb (87) tfarmemw (87)

11.2 Predefined variables

11.2.1 tdosmem based variables

Declaration:
```
VAR    dmemb : tdosmemb;
      dmemw : tdosmemw;
      dmeml, dmemd : tdosmeml;
```

Description: These variables can be used to access dos-memory, don't forget to initialise the classes first.

See also: tfarmem based variables (88) tdosmemb (86) tdosmemw (86) tdosmeml (87)

11.2.2 tfarmem based variables

Declaration:
```
VAR    dmemb : tfarmemb;
      dmemw : tfarmemw;
      dmeml, dmemd : tfarmeml;
```

Description: These variables can be used to access a different segment, don't forget to initialise the classes first.

Also don't forget that %gs isn't saved, which can be a problem with some Linear Frame Buffer VESA units.

See also: tdosmem based variables (88) tfarmemb (87) tfarmemw (87) tfarmeml (88)

Chapter 12

The Memory Unit

`written by Thomas Schatzl`

12.1 FEATURES

- Fast memory transfers by using the FPU, speed increase of up to 100on Intel Pentium systems
- Fast memory set procedures
- Replaces the go32 move* and fillchar* functions with high speed assembler procedures completely
- 16 boolean operations between two memory regions or memory and a single value possible
- Boolean operations match Mircosoft ROP order
- Automatically uses MMX extensions if available
- Writes information to stderr if DEBUG is defind

12.2 BACKGROUND

This unit was done because I need fast memory copy routines and boolean bit operation for a greater project of mine, a graphics library which uses the 2D acceleration features of most graphics chips nowadays. Some procedures are needed to support all functions (bitblt, HW memory copies..) in software for chipsets which don't support HW acceleration or miss one or another feature or are simply older.

On the other hand I wanted to know what comes out if you take advantage of modern chipsets available instruction sets. And it seems it was worth the effort, because there's a speed gain of up to 400instructions to copy memory, or using Intel's MMX extensions for boolean operations on my P200 MMX.

12.3 SYSTEM REQUIREMENTS

This unit requires an IBM PC or compatible with an 80386 or higher processor.

Table 12.1: Mem_Op Operation modes

Enumeration name	Boolean bit operation result	Description Note
MEM_CLEAR	dst = 0	zero out destination
MEM_NOT_DSTORSRC	dst = not (dst or src)	
MEM_DST_AND_NOTSRC	dst = dst and (not src)	
MEM_NOTSRC	dst = not src	invert source and copy
MEM_SRC_AND_NOTDST	dst = src and (not dst)	
MEM_NOTDST	dst = not dst	invert destination
MEM_DST_XOR_SRC	dst = dst xor src	
MEM_NOT_DSTANDSRC	dst = not (dst and src)	
MEM_DST_AND_SRC	dst = dst and src	
MEM_NOT_DSTXORSRC	dst = not (dst xor src)	
MEM_DST	dst = dst	does nothing
MEM_DST_OR_NOTSRC	dst = dst or (not src)	
MEM_SRC	dst = src	copy / fill
MEM_NOTDST_OR_SRC	dst = (not dst) or src	
MEM_DST_OR_SRC	dst = dst or src	
MEM_SET	dst = 1	set destination

12.4 PROGRAMMING LANGUAGE

This Memory unit can be compiled with FPC Pascal (32-bit protected mode freeware DOS compiler) and a GNU assembler version of 2.9.1 or higher.

Supports following extenders:

- GO32V1
- GO32V2
- PMODE D/J

Needs the following switches enabled:

- none

12.5 Types of the memory unit

12.5.1 Mem_Op (enumeration)

Describes the 16 possible boolean operations between two values

`Description`

This enumeration describes the 16 possible boolean operations between two bits (in this case src and dst) using the and, or, xor and not operators.

Table table (12.1) lists the possible operation modes.

The result is archieved by combining src and dst values bit by bit.

Example 1:

src = %10001
dst = %01011

Table 12.2: Four different basic operations

AND			OR			XOR			NOT	
A	B	result	A	B	result	A	B	result		
0	0	0	0	0	0	0	0	0	A	result
1	0	0	1	0	1	1	0	1	1	0
0	1	0	0	1	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0		

selected operation : MEM_DST_OR_NOTSRC

dst = dst or (not src)
= %01011 or (not %10001)
= %01011 or %01111
= %01111

Example 2:

src = %10001
dst = %01001

selected operation : MEM_CLEAR

dst = 0
= %00000

In this case the src and dst memory contents are not important at all.

The next table shows the results of the four different basic operations according to given expressions A and B.

Example:

A = %10100
B = %01001
operation : XOR
result = %10100 xor %01001 = %10101

12.5.2 DWORD

This type defines a 32 bit unsigned integer.

```
type DWord = Cardinal;
```

12.6 Memory Functions

12.6.1 memcpy

Declaration: `PROCEDURE memcpy (var src, dst; size :  DWord);`

Description: Copies size bytes between src and dst. No range checking is performed and it can handle overlapping memory areas correclty. A speed gain of up to 100floating point registers for copying.

See also: seg_memcpy (92)

12.6.2 memset

Declaration:
```
procedure memsetB (var x; size : DWord; value : Byte);
procedure memsetW (var x; size : DWord; value : Word);
procedure memsetD (var x; size : DWord; value : DWord);
```

Description: Sets size bytes beginning from x to the value value. No range checking is performed, so the use of the sizeof() operator is recommened. Uses 64 bit FPU registers for fast memory fill. The difference between the three routines is the size of the value argument. The one fills the memory with a Byte value, the other with a Word value and the last with a DWord value.

See also: seg_memsetB (93) seg_memsetW (93) seg_memsetD (93)

12.6.3 memchange

Declaration:
```
procedure memchange (var src, dst; size : DWord; op : Mem_Op);
```

Description: Does a boolean operation between two memory areas

Combines size bytes of src and dst via the boolean operation op and stores the result in dst. The boolean operation is applied on a bit to bit basis. Automatically uses MMX extensions if available for speed enhancement. No range checking is performed, so it is recommened to use the sizeof() operator. When the src and dst memory area overlap the result is undefined.

See also: Mem_Op (12.5.1) seg_memchange (93)

12.6.4 memchangeValue

Declaration:
```
procedure memchangeValueB (var x; size : DWord; value : Byte; op : Mem_Op);
procedure memchangeValueW (var x; size : DWord; value : Word; op : Mem_Op);
procedure memchangeValueD (var x; size : DWord; value : DWord; op :
Mem_Op);
```

Description: Combines size bytes of value and x via the selected boolean operation in op and stores the result in x. Automatically uses MMX extensions when available. No range checking is performed, so the use of the sizeof() operator is recommened. The difference between the three procedures is the size of the value, either Byte, Word or DWord.

See also: Mem_Op (12.5.1) seg_memchangeValueB (93) seg_memchangeValueW (93) seg_memchangeValueD (93)

12.6.5 seg_memcpy

Declaration:
```
procedure seg_memcpy (srcsel : Word; srcofs : DWord; dstsel : Word;
dstofs: Dword; size : DWord);
```

Description: Copies bytes from src to dst across selector boundaries.

This procedure does the same as memcpy, except that the src and dst may reside in different memory segments or outside the DS selector range.

See also: memcpy (91)

12.6.6 seg_memset

Declaration: `procedure seg_memsetB (sel : Word; ofs : DWord; size : DWord; value : Byte);`
`procedure seg_memsetW (sel : Word; ofs : DWord; size : DWord; value : Word);`
`procedure seg_memsetD (sel : Word; ofs : DWord; size : DWord; value : DWord);`

Description: Sets size bytes across selector boundaries.

This procedure does the same as memset*, except that the memory may reside outside the DS selector range.

See also: memsetB (92) memsetW (92) memsetD (92)

z

12.6.7 seg_memchange

Declaration: `procedure seg_memchange (srcsel : Word; srcofs : DWord; dstsel : Word; dstofs : DWord; size : DWord);`

Description: Combines two memory areas by a boolean operation which aren't in the DS selector range

This procedure does the same as memchange, except that the memory areas may reside outside the DS selector range or be in different memory areas.

See also: Mem_Op (12.5.1) memchange (92)

12.6.8 seg_memchangeValue

Declaration: `procedure seg_memchangeValueB (sel : Word; ofs : DWord; size : DWord; value: Byte; op : Mem_Op);`
`procedure seg_memchangeValueW (sel : Word; ofs : DWord; size : DWord; value: Word; op : Mem_Op);`
`procedure seg_memchangeValueD (sel : Word; ofs : DWord; size : DWord; value: DWord; op : Mem_Op);`

Description: Combines a memory area with a value by a boolean operation which is outside the DS selector range.

This procedure does the same as memchangeValue*, except that the memory area may reside outside the DS selector range.

See also: Mem_Op (12.5.1) memchangeValueB (92) memchangeValueW (92) memchangeValueD (92)

If you have any questions or feedback then e-mail me at tom_at_work@geocities.com.

Chapter 13

DPMI Unit (DPMI 0.9)

This documentation contains information about protected mode, the DPMI interface and the DPMI unit itself. This should be a brief introduction into the basics of protected mode programming in general and with FPC using this unit.

The main points which are discussed within this chapter are :

- Protected Mode (section 13.1)
- The DPMI Interface (section 13.2 itself)
- The DPMI Unit (section 13.3)
- DPMI unit function descriptions (section 13.4)

The original DPMI document was compiled by Thomas Schatzl on 1 January 1999.
For questions, suggestions, improvements or other things, mail at tom_at_work@geocities.com.

The Tex conversion was done by Marco van de Voort (MarcoV@stack.nl)

13.1 Protected mode

Protected mode was designed to fit the needs of modern multitasking operating systems. Multitasking means that one or more actions (tasks) are seemingly done at the same time. A good example for this is printing a document in the background while doing something else.

These demands for CPUs are roughly:

- Protection of different tasks and operating systems against writes to invalid memory areas
- Support at task-switching, preferably for saving and restoring of the task state
- Virtual memory support
- Privilegizing of the operating system for instructions

From the above four requirements the first one is the most striking (for GO32V2 programmers), because that means that memory access like in real mode would exactly against the rule. In real mode the programmer could access any memory cell below the 1 MB boundary by simply loading a value between $0000 and $FFFF into any segment register to overwrite important parts of the operating system like the interrupt vector table, parts of COMMAND.COM, and other things which surely cause the CPU to reset.

In protected mode you may get in big troubles, even when loading a random value of a segment address into a segment register, because this causes the processor to stop executing the program. This doesn't reset or crash the processor, but he simply calls an exception (an interrupt) which returns control to the underlying operating system again, which in term simply terminates the program. It's that easy in protected mode.

This is only possible because segment addresses aren't segment addresses anymore, but (segment-)selectors. Selectors are indexes into a table of segment descriptors which describe a single memory area. The CPU gets the base address of the memory segment. Finally he adds the offset address from the pointer to this value to get the real memory address of e.g. a variable. (No more multiply of the segment by 16 and then adding the offset like in real mode, hence memory segments can start at any address)

A descriptor holds more than simply the base address of the segment, but several additional information. These are the base address, the length of the segment and finally some flags. (Please refer to a good book / documentation for further reference, because that's all that a protected mode programmer needs to know as long as he doesn't want to make his own operating system....)

This is where a DPMI extender comes into play.

13.2 The DPMI Interface

The DOS Protected Mode Interface (DPMI) was primarily defined to allow DOS programs access to the extended memory area (memory ¿ 1MB) while maintaining system protection. DPMI defines a subset of DOS and BIOS calls which can be made by protected mode DOS programs. It also provides a new interface via Int 31h that protected mode programs use to allocate memory, modify descriptors, call real mode software, etc.
Such a program is commonly called as a DPMI host.

13.3 The DPMI unit

This unit provides an interface to the various functions a DPMI host provides for its application with the FPC compiler. Additionally I tried to add several functions to simplify handling with selectors and some other useful procedures and features which are missing in the GO32 unit.

Several issues of the DPMI API are handled by this unit:

- Error handling (99)
- Initialization services (99)
- LDT descriptor management (100)
- DOS memory management (107)
- Interrupt services (108)

- Translation services (113)
- Get Version (118)
- Memory management services (105)
- Page locking services (115)
- Demand paging performance tuning services (117)
- Physical address mapping (106)
- Virtual interrupt state functions (111)

In addition to these, the unit DPMI provides several other functions

- Common used function combinations (”Time savers”) (118)
- Segment register access (121)
- Port access (124)
- Hardware interrupts handling (125)
- Transfer buffer access (126)
- ”Near Pointer”-handling (127)
- Fast memory transfer, set and change functions (using unit Memory chapter 12)
- Different mem[]-”arrays” suited for protected mode (using unit Farmem chapter 11)

The documentation of unit Memory is ready at the moment of writing and should be included in this document distribution. The documentation of unit Farmem is not ready yet.

For people who want to port their code from the GO32 unit to the DPMI unit, look at table (13.4) (Porting Go32 code to DPMI)

13.4 DPMI unit function descriptions

13.5 Types and Constants

13.5.1 Type : Descriptor

```
Descriptor = array[0..7] of Byte;
```

This array holds the contents of a descriptor. Since descriptor handling is normally not done by the common programmer, I didn’t go into more detail. For further reference what this array exactly contains, please consult a good protected mode documentation.

See also LDT management services (100)

13.5.2 Type: Registers

```
type  Registers = record
                   case integer of
                    0 : (edi, esi, ebp, res1, ebx, edx, ecx, eax : DWord;
                           Flags, es, ds, fs, gs, ip, cs, sp, ss : Word);
                    1 : (di, di2, si, si2, bp, bp2, res2, res3,
                           bx, bx2, dx, dx2, cx, cx2, ax, ax2 : Word);
                    2 : (res4 : array[1..4] of DWord;
                           bl, bh, bl2, bh2, dl, dh, dl2, dh2,
                           cl, ch, cl2, ch2, al, ah, al2, ah2 : Byte);
                  end;
```

This data structure contains the values which must be passed to or are returned by an interrupt call or a real mode callback structure. See at Translation services (113) for further reference about the usage of this structure

13.5.3 Type: Flags constants

```
const
 fCarry = $0001;
 fParity = $0004;
 fAuxiliary = $0010;
 fZero = $0040;
 fSign = $0080;
 fTrap = $0100;
 fInterrupt = $0200;
 fDirection = $0400;
 fOverflow = $0800;
```

These constants define the bit location of the various flags within the flags-entry of the 'register' data structure. See Type Registers (97).

13.5.4 Type: PM_Addr

```
type    pm_Addr = record
                   offset : DWord;
                   selector : Word;
                   end;
```

This is the full definition of a single memory address in protected mode. It is 48 bit in size, since a single memory segment is still 16 bit and it can be up to 32 bits in size. See Interrupt services (108) or Translation services (113) for applications.

13.5.5 Type: RM_Addr

```
type
     rm_Addr = record
     offset : Word;
     segment : Word;
     end;
```

Definition of a real mode address in segment:offset notation. See Translation services (113) or Interrupt services (108) for applications.

Table 13.1: Description of MemInfoBuf

Field identifier	Description
largest_available_free_block	Largest available free block in bytes
max_unlocked_pages	Maximum unlocked page allocation
max_lockable_pages	Maximum locked page allocation
linear_address_space_size	Linear address space size in pages
number_of_unlocked_pages	Total number of unlocked pages
number_of_free_pages	Number of free pages
number_of_physical_pages	Total number of physical pages
free_linear_address_space	Free linear address space in pages
size_of_paging_file	Size of paging file/partition in pages
reserved	Reserved

13.5.6 Type: MemInfoBuf

```
type
     MemInfoBuf = record
                  largest_available_free_block : Longint;
                  max_unlocked_pages           : Longint;
                  max_lockable_pages           : Longint;
                  linear_address_space_size    : Longint;
                  number_of_unlocked_pages     : Longint;
                  number_of_free_pages         : Longint;
                  number_of_physical_pages     : Longint;
                  free_linear_address_space    : Longint;
                  size_of_paging_file          : Longint;
                  reserved                     : array[0..2] of Longint;
                  end;
```

The information block about memory used/allocated of the program which is returned by dpmi_get_free_memory_information (105) only. See table (13.1) for a short description of the fields.

Only the first entry is guaranteed to contain a valid value, the others contain 1 ($FFFFFFFF) if invalid. To get the page size in bytes, look at dpmi_get_page_size (116).

13.5.7 Variable: Dpmi_Error

```
var dpmi_error : DWord;
```

Contains the last error number occured by a DPMI call. See Errorhandling (99) too.

13.6 DPMI functions and procedures

13.6.1 Error Handling

Since any of the DPMI calls can fail, I decided to let the programmer install its own error handler which is called automatically at a DPMI call failure. This is done via a special error routine ('handler') which is called everytime an error occurs. This handler can be redefined by the programmer to suit its own purposes. The error handler is a simple procedure which gets the last error code as a dword as an argument.

Such an error code consists of three parts: First comes the warning/fatal bits, then a unit identifier to get to know which unit caused the error and last the error code itself.

The fatal/warning bits are the upper 2 bits of the parameter dword. They may be ignored by the error handler (because the caller decides the type of the error), but it's generally a good idea to halt on a fatal error.
Then comes a 10 bit sized unit identifier, which tells the error handler at which unit the error happened.
And last a 20 bit sized error code, which can be output for reference.

Additionally *every* dpmi_xxxx function returns a boolean result which indicates success or fail, in the case you don't want to care about the dpmi_error variable every time (and added a dummy error handler which does completely nothing).

Btw, error handling will be changed soon (too unflexible now), to something like done in the API unit, so the error codes in the function descriptions aren't entered into the proper row. See table 2 where the error codes for all functions which return errors are listed with their associated error code.

13.6.2 dpmi_set_error_handler

Declaration: `procedure dpmi_set_error_handler(handler : error_proc);`

Description: Sets a new error handler for the DPMI unit

Parameters: handler the address of the new error handler procedure

Return value: none

Error code: none

Notes: The standard error handler updates the dpmi_error variable, writes a message to stdout, and triggers an RTE 216 (General Protection Fault) if the error was a fatal one.

See also: Dpmi_Error (98), Error handling (99)), table (13.2)

13.6.3 Initialization services

These function deal with detecting the current mode the cpu runs under

13.6.4 dpmi_get_cpu_mode

Declaration: `function dpmi_get_cpu_mode(var in_pm : Boolean) : Boolean;`

Description: Returns information about the current CPU mode

Parameters: none

Return value: in_pm true if running under protected mode, false if not

Error code: none

Notes: GO32V2 applications generally don't need to care about this.

See also: Initialization services,(99),

13.6.5 LDT management services

The LDT (local descriptor table) management services provide several interfaces to allocate, free, resize, lock and unlock protected mode descriptors for the current task.

13.6.6 dpmi_allocate_ldt_descriptors

Declaration: `function dpmi_allocate_ldt_descriptors(number : Word; var basesel : Word) : Boolean;`

Description: Allocates one or more descriptors from the local task's LDT. The descriptors allocated must be initializd by the application.

Parameters: number number of selectors to allocate

Return value: basesel the base selector to the array of selectors allocated

Error code: $0000

Notes: If more than one descriptor was requested, basesel contains the first of a contiguous array of descriptors. You should add the value returned by dpmi_get_next_selector_increment_value() to get to the next selector in the array.

See also: LDT management services,(100), dpmifreeldtdescriptor (100)

dpmisetsegmentbaseaddress (102), dpmigetsegmentbaseaddress (101), dpmisetsegmentlimit (102) dpmigetsegmentlimit (102), dpmisegmenttodescriptor (101), dpmigetnextselectorincrementvalue (101) createselector (118), freeselector (119), mapphysicalmemory (119) table (13.2)

13.6.7 dpmi_free_ldt_descriptor

Declaration: `function dpmi_free_ldt_descriptor(sel : Word) : Boolean;`

Description: This function is used to free previously allocated selectors by dpmi_allocate_ldt_descriptors()

Parameters: sel the selector to deallocate

Return value: none

Error code: $0001

Notes: Arrays of selectors must be freed individually by calling this function for each single selector. You may use the free_selector() function instead of this too.

See also: LDT management services,(100), dpmiallocateldtdescriptors (100) freeselector (119), table (13.2)

13.6.8 dpmi_segment_to_descriptor

Declaration: `function dpmi_segment_to_descriptor(seg : Word; var sel : Word) : Boolean;`

Description: Converts a real mode segment into a descriptor which can be accessed by protected mode programs.

Parameters: seg real mode segment address

Return value: sel selector mapped to real mode address

Error code: $0002

Notes: The selectors limit is always set to $FFFF (64k). Multiple calls to this function with the same segment address will return the same selector. For this reason, selectors created by this selector should never be freed or modified. Use this function sparingly. If you want to examine different real mode segments it is better to allocate a single selector by dpmi_allocate_ldt_descriptors() and change its base address via dpmi_set_segment_base_address() accordingly.

See also: LDT management services,(100), dpmiallocateldtdescriptors (100) dpmisetsegmentlimit (102), dpmisetsegmentbaseaddress (102), table (13.2)

13.6.9 dpmi_get_next_selector_increment_value

Declaration: `function dpmi_get_next_selector_increment_value(var incval : Word) : Boolean;`

Description: Some functions (like allocate_ldt_descriptors()) can return more than one descriptor. You must call this function to determine the value which must be added to a selector to gain access to the next descriptor in the array.

Parameters: none

Return value: incval value to add to get to next selector

Error code: $0003

Notes: The returned value will be a power of two, but don't make assumptions about the value this function will return.

See also: LDT management services,(100), dpmiallocateldtdescriptors (100) table (13.2)

13.6.10 dpmi_get_segment_base_address

Declaration: `function dpmi_get_segment_base_address(sel : Word; var baseaddr : DWord) : Boolean;`

Description: This function returns the 32bit linear base address of the specified segment.

Parameters: sel selector

Return value: baseaddr 32 bit linear base address of segment

Error code: $0006

Notes: This function fails if sel is invalid.

See also: LDT management services,(100), dpmiallocateldtdescriptors (100) dpmisetsegmentbaseaddress (102), dpmisetsegmentlimit (102), dpmigetsegmentlimit (102) getlinearaddress (119), table (13.2)

13.6.11 dpmi_set_segment_base_address

Declaration: `function dpmi_set_segment_base_address(sel : Word; baseaddr : DWord) : Boolean;`

Description: This function changes the 32 bit linear address of the specified segment

Parameters: sel segment to change base address baseaddr new 32 bit base address

Return value: none

Error code: $0007

Notes: This function fails if sel is invalid. You should only modify descriptors that were allocated via dpmi_allocate_ldt_descriptors() before.

See also: LDT management services,(100), dpmiallocateldtdescriptors (100) dpmigetsegmentbaseaddress (101), dpmisetsegmentlimit (102), dpmigetsegmentlimit (102) createselector (118), changeselector (119), table (13.2)

13.6.12 dpmi_get_segment_limit

Declaration: `function dpmi_get_segment_limit(sel : Word; var limit : DWord) : Boolean;`

Description: Returns the 32 bit limit of the specified segment

Parameters: sel selector

Return value: limit limit of the specified segment in bytes 1

Error code: See table about Error codes

Notes: This function will fail if the supplied selector is invalid.

See also: LDT management services,(100), dpmisetsegmentlimit (102) dpmigetsegmentbaseaddress (101), dpmisetsegmentbaseaddress (102), table (13.2)

13.6.13 dpmi_set_segment_limit

Declaration: `function dpmi_set_segment_limit(sel : Word; limit : DWord) : Boolean;`

Description: This function sets the limit for the specified segment

Parameters: sel selector limit new 32 bit limit of segment in bytes 1

Return value: none

Error code: $0008

Notes: This function will fail when the specified selector is invalid or the requested limit couldn't be set. Segment limits greater than 1 MB must be page aligned, that means, the lower 12 bits of the limit field must have the lower 12 bits set. Your program should only modify descriptors that were previously allocated via the dpmi_allocate_ldt_descriptors() function.

See also: LDT management services,(100), dpmiallocateldtdescriptors (100) dpmigetsegmentlimit (102), dpmisetsegmentbaseaddress (102), dpmigetsegmentbaseaddress (101) changeselector (119), table (13.2)

13.6.14 dpmi_get_descriptor_access_rights

Declaration: `function dpmi_get_descriptor_access_rights(sel : Word; var rights : Word) : Boolean;`

Description: Returns the access rights and type field of the specified descriptor.

Parameters: sel selector

Return value: rights access rights / type field of the descriptor

Error code: See notes in Error codes table.

Notes: This function fails if the specified selector is invalid. See a DPMI specification for the exact contents of the rights variable.

See also: LDT management services,(100), dpmisetdescriptoraccessrights (103) dpmigetdescriptor (104), dpmisetdescriptor (104), dpmiallocatespecificdescriptor (104) table (13.2)

13.6.15 dpmi_set_descriptor_access_rights

Declaration: `function dpmi_set_descriptor_access_rights(sel : Word; rights : Word) : Boolean;`

Description: This function allows protected mode programs to modify the access rights field of a descriptor

Parameters: sel selector rights new access rights / type

Return value: none

Error code: $0009

Notes: This function will fail if the selector specified is invalid. Your program should only change the access rights / type of descriptors allocated via the dpmi_allocate_ldt_descriptors() function.

See also: LDT management services,(100), dpmigetdescriptoraccessrights (103) dpmigetdescriptor (104), dpmisetdescriptor (104), dpmiallocatespecificdescriptor (104) table (13.2)

13.6.16 dpmi_create_code_segment_alias_descriptor

Declaration: `function dpmi_create_code_segment_alias_descriptor(codesel : Word; var sel : Word) : Boolean;`

Description: This function will create a data descriptor that has the same base and limit as the specified code segment descriptor.

Parameters: codesel code segment selector

Return value: sel new data segment selector

Error code: $000A

Notes: This function fails if the specified selector is invalid or not a code segment selector. You have to use the dpmi_free_ldt_descriptor() function to free such a descriptor afterwards. The new data segment descriptor will not track changes from the code segment descriptor. With FPC you don't need this function, because you can write to the code segment too (because it is automatically set as read/writeable at startup)

See also: LDT management services,(100), dpmigetdescriptoraccessrights (103) dpmisetdescriptoraccessrights (103), table (13.2)

13.6.17 dpmi_get_descriptor

Declaration: `function dpmi_get_descriptor(sel : Word; var descr : Descriptor) : Boolean;`

Description: This function copies the descriptor table entry for a specified descriptor into a buffer

Parameters: sel selector

Return value: descr 8 byte buffer holding the descriptor values

Error code: $000B

Notes: This function will fail if the selector specified is invalid or unallocated.

See also: LDT management services,(100), dpmisetdescriptor (104) dpmiallocatespecificdescriptor (104), dpmisetdescriptoraccessrights (103) dpmigetdescriptoraccessrights (103), table (13.2)

13.6.18 dpmi_set_descriptor

Declaration: `function dpmi_set_descriptor(sel : Word; descr : Descriptor) : Boolean;`

Description: This function copies an 8 byte buffer into the LDT entry for a specified descriptor

Parameters: sel selector
desc 8 byte buffer containing descriptor

Return value: none

Error code: $000C

Notes: This function will fail if the selector specified is invalid. You should only modify descriptors allocated by the dpmi_allocate_ldt_descriptors() function. For a complete description of the contents of the 8 byte buffer, please refer to a DPMI specification.

See also: LDT management services,(100), dpmigetdescriptor (104) dpmiallocatespecificdescriptor (104), table (13.2)

13.6.19 dpmi_allocate_specific_descriptor

Declaration: `function dpmi_allocate_specific_descriptor(sel : Word) : Boolean;`

Description: This function attempts to allocate a specific LDT descriptor.

Parameters: sel selector

Return value: none

Error code: $000D

Notes: This function will fail if the specified selector is in use or not an LDT selector. You need to use the dpmi_free_ldt_descriptor() function to free this descriptor again.

See also: LDT management services,(100), dpmigetdescriptor (104) dpmisetdescriptor (104), table (13.2)

13.6.20 Memory management services

These functions are provided to allocate memory in linear address space. This should normally be of no concern to the standard FPC programmer, because FPC automatically 'grows' its heap.

13.6.21 dpmi_get_free_memory_information

Declaration: `function dpmi_get_free_memory_information(var mem : meminfobuf) : Boolean;`

Description: This function is provided so that protected mode applications can determine how much memory is available. Under DPMI implementations that support virtual memory, it is important to consider issues such as the amount of available physical memory.

Parameters: mem 30 byte buffer

Return value: mem buffer filled out with memory information

Error code: $0002

Notes: To determine the size of pages, call the dpmi_get_page_size() function.

See also: Memory management services,(105), dpmigetpagesize (116), table (13.2)

13.6.22 dpmi_allocate_memory_block

Declaration: `function dpmi_allocate_memory_block(size : DWord; var linearaddr : DWord; var blockhandle : DWord) : Boolean;`

Description: This function allocates and commits linear memory.

Parameters: size size of requested memory block in bytes

Return value: linearaddr linear address of allocated memory block blockhandle memory block handle (used to resize and free memory)

Error code: $0002

Notes: This function does not allocate any selectors for the memory block. It is up to the application to allocate and initialize selectors needed for access. The block is allocated unlocked. Allocations will be page granular, this means that an allocation of $1001 bytes will result in allocation of $2000 bytes. Therefore it's best to always allocate memory in multiplies of 4K.

See also: Memory management services,(105), dpmifreememoryblock (106), dpmiresizememoryblock (106), table (13.2)

13.6.23 dpmi_free_memory_block

Declaration: `function dpmi_free_memory_block(blockhandle : DWord) : Boolean;`

Description: This function frees a memory block that was allocated through the dpmi_allocate_memory_block() function.

Parameters: blockhandle handle of memory block to free

Return value: none

Error code: $0002

Notes: Your programs must also free selectors that it allocated to access the memory.

See also: Memory management services,(105), dpmiallocatememoryblock (105), table (13.2)

13.6.24 dpmi_resize_memory_block

Declaration: `function dpmi_resize_memory_block(newsize : DWord; var blockhandle : DWord; var linearaddr : DWord) : Boolean;`

Description: This function changes the size of a memory block that was allocated through the dpmi_allocate_memory_block() function.

Parameters: newsize new size of memory block in bytes blockhandle handle of memory block to resize

Return value: linearaddr new linear address of memory block blockhandle new blockhandle of memory block

Error code: $0002

Notes: This function may change the linear address of the memory block and its handle. Therefore, you'll need to update any selectors that point to the block after resizing it. You must use the new handle instead of the old one.

See also: Memory management services,(105), dpmiallocatememoryblock (105), dpmifreememoryblock (106), table (13.2)

13.6.25 Physical address mapping

Memory mapped devices such as network or display adapters sometimes have memory mapped at physical addresses beyond the normal 1 Mb of realmode addressable memory space. This service can be used to convert physical addresses of such mapped memory into linear addresses. This 32 bit linear address then can be used to access this memory.

13.6.26 dpmi_physical_address_mapping

Declaration: `function dpmi_physical_address_mapping(physaddr : DWord; size : DWord; var linearaddr : DWord) : Boolean;`

Description: Converts physical memory addresses of devices into 32 bit linear addresses

Parameters: physaddr physical address of memory range size size of memory region in bytes 1

Return value: linearaddr linear address of device memory

Error code:

Notes: The returned linear address can be used to access the devices' memory. The application must create a valid descriptor to this memory before accessing it. Do not use this service to access memory that is mapped into the first megabyte of address space. (The physical address equals the linear address in this memory area only)

See also: Physical address mapping,(106), mapphysicalmemory (119), table (13.2)

13.6.27 DOS memory management

Some programs require the ability to allocate memory in the real mode addressable ¡ 1 mb region. These following services allow protected mode applications to allocate and free memory that is directly addressable by real mode software such as networks and DOS device drivers. Often, this memory is used in conjunction with the translation services to call real mode software that is not directly supported by DPMI.

For easier access to this region the FPC team decided to automatically load the memory region. So you must make sure that this segment register never changes in your code, because some functions rely on this. Additionally they automatically provide a preallocated transfer buffer to be available for temporary use (e.g. as long as you don't call certain FPC functions).

Some tips about address calculation follow:

To obtain the linear address (offset) from the DOS memory selector, you need to calculate the address like it was in real mode. This linear address then can be used to copy it to protected mode heap.

(linear_address := segment * 16 + offset)

In reverse, to generate a segment:offset address out of a linear address do the following:

segment := linear_address div 16; offset := linear_address and 15;

This can be useful when you need to pass the segment:offset address of e.g. the transfer buffer to a real mode interrupt.

13.6.28 dpmi_allocate_DOS_memory_block

Declaration: `function dpmi_allocate_DOS_memory_block(size : DWord; var realaddr : rm_Addr; var sel : Word) : Boolean;`
or `function dpmi_allocate_DOS_memory_block(size : DWord; var realseg : Word; var sel : Word) : Boolean;`

Description: This function will allocate a block of memory from the DOS memory pool. It returns both the real mode memory segment and a descriptor which can be used by protected mode applications.

Parameters: size size of requested DOS memory block in bytes

Return value: realaddr DOS real mode segment:offset address of the allocated memory block realseg DOS real mode segment of the allocated memory block sel protected mode selector for allocated block

Error code: $0002

Notes: Your program should never modify or deallocate any descriptors allocated by this function. The dpmi_free_DOS_memory_block() function will automatically deallocate the descriptors

See also: DOS memory managment,(107), dpmifreedosmemoryblock (108) dpmiresizeDOSmemoryblock (108), table (13.2)

13.6.29 dpmi_free_DOS_memory_block

Declaration: `function dpmi_free_DOS_memory_block(sel : Word) : Boolean;`

Description: This function frees memory that was previously allocated by the dpmi_allocate_DOS_memory_block() function.

Parameters: sel selector

Return value: none

Error code: $0002

Notes: The descriptor allocated for the memory block is automatically freed and therefore should not be accessed once the freed by this function.

See also: DOS memory managment,(107), dpmiallocatedosmemoryblock (107) table (13.2)

13.6.30 dpmi_resize_DOS_memory_block

Declaration: `function dpmi_resize_DOS_memory_block(newsize : DWord; sel : Word) : Boolean;`

Description: This function is used to grow or shrink a memory block that was previously allocated by the dpmi_allocated_DOS_memory_block() function.

Parameters: newsize new block size in bytes sel selector of block to modify

Return value: none

Error code: $0002

Notes: Growing a DOS memory block is often likely to fail because other DOS memory allocations will prevent increasing the size of the block. Therefore, this function is usually only used to shrink a block

See also: DOS memory managment,(107), dpmiallocatedosmemoryblock (107) dpmifreedosmemoryblock (108), table (13.2)

13.6.31 Interrupt services

These services allow protected mode programs to hook to interrupts and intercept processor exceptions.

All interrupts from hardware (like keyboard, or timer) will always be reflected to the protected mode handler first. If the protected mode handler jumps or calls the previous interrupt handler, then the interrupt will be reflected to real mode.

As in real mode, interrupt procedures can either service the interrupt via iret or they can chain to the next interrupt handler in an interrupt chain. The final handler for all protected mode handlers will reflect the interrupt to real mode (e.g. the real mode interrupt gets executed). When an interrupt is reflected to real mode protected to real mode. The segment registers contain undefined values. The DPMI host automatically provides a real mode stack for interrupts that are reflected to real mode.

Hardware Interrupts

The interrupt controllers are mapped to the system's default interrupts (e.g. the master interrupt controller has a base interrupt of $8 and the slave controller has a base of $70). Hardware interrupt procedures and all of their data must reside in locked memory. All that memory touched by hardware interrupt hooks must be locked. The handler will always be called on a locked stack. As in real mode, hardware interrupts are called with interrupts disabled. Since iret doesn't restore the interrupt flag, the handler must enable interrupts via a sti instruction, else interrupts will remain disabled.

Software interrupts

Most software interrupts executed in real mode will not be reflected to protected mode interrupt hooks. However, there are some exceptions: Int $1C (BIOS timer interrupt), Int $23 (DOS Ctrl+C interrupt), Int $24 (DOS critical error interrupt).

13.6.32 dpmi_get_rm_interrupt

Declaration: `function dpmi_get_rm_interrupt(number : Byte; var addr : rm_Addr) : Boolean;`

Description: This function returns the value of the current task's real mode interrupt vector for the specified interrupt

Parameters: number interrupt number

Return value: rm_Addr address of real mode interrupt vector

Error code:

Notes: The returned address is a real mode segment:offset address, not a protected mode one.

See also: Interrupt services,(108), dpmisetrminterrupt (109) dpmigetpminterrupt (111), dpmisetpminterrupt (111), dpmigetexceptionhandler (110) dpmisetexceptionhandler (110), dpmisimulaterminterrupt (113), dpmiallocatermcallback (114) intr (120), realintr (120), rm_addr (97), table (13.2)

13.6.33 dpmi_set_rm_interrupt

Declaration: `function dpmi_set_rm_interrupt(number : Byte; addr : rm_Addr) : Boolean;`

Description: This function sets the value of the current task's real mode vector for the specified interrupt.

Parameters: number interrupt number to set addr address of real mode interrupt

Return value: none

Error code: $0002

Notes: The address passed to this function must be a real mode segment:offset address, not a protected mode selector:offset address. This means that the code for the interrupt handler must either be in DOS addressable memory or you must use a real mode callback address. Refer to dpmi_allocate_DOS_memory() on allocating memory below 1 Mb. Information on real mode callbacks can be found at the Translation services section.

See also: Interrupt services,(108), dpmigetrminterrupt (109) dpmigetpminterrupt (111), dpmisetpminterrupt (111), dpmigetexceptionhandler (110) dpmisetexceptionhandler (110), dpmisimulaterminterrupt (113), dpmiallocatermcallback (114) intr (120), realintr (120), rm_addr (97), table (13.2)

13.6.34 dpmi_get_exception_handler

Declaration: `function dpmi_get_exception_handler(number : Byte; var addr : pm_Addr) : Boolean;`

Description: This function returns the address of the current protected mode exception handler for the specified exception number.

Parameters: number exception number

Return value: addr selector:offset of the exception

Error code: $0002

Notes: This function fails if the number passed was invalid.

See also: Interrupt services,(108), dpmisetexceptionhandler (110) dpmigetrminterrupt (109), dpmisetrminterrupt (109), dpmigetpminterrupt (111) dpmisetpminterrupt (111), dpmiallocatermcallback (114), dpmisimulaterminterrupt (113) intr (120), realintr (120), pm_addr (97), table (13.2)

13.6.35 dpmi_set_exception_handler

Declaration: `function dpmi_set_exception_handler(number : Byte; addr : pm_Addr) : Boolean;`

Description: This function allows protected mode programs to intercept processor exceptions that are not handled by the DPMI environment. Programs may wish to handle exceptions such as protection faults which would otherwise generate a fatal error.

Parameters: number exception/fault number ($00$1F) pm_addr selector:offset address of exception handler

Return value: none

Error code: $0002

Notes: Every exception is first examined by the protected mode operating system. If it cannot handle it, it then reflects it through the protected mode exception handler chain. The final handler in the chain may either reflect the exception as an interrupt or terminate the program. This function fails if the exception/fault number passed is invalid. For further reference on how to deal with exceptions (and setting up a handler), please consult a DPMI reference.

See also: 2 Interrupt services,(108), dpmigetexceptionhandler (110) dpmigetrminterrupt (109), dpmisetrminterrupt (109), dpmigetpminterrupt (111) dpmisetpminterrupt (111), dpmiallocatermcallback (114), dpmisimulaterminterrupt (113) intr (120), realintr (120), pm_addr (97), table (13.2)

13.6.36 dpmi_get_pm_interrupt

Declaration: `function dpmi_get_pm_interrupt(number : Byte; var addr : pm_Addr) : Boolean;`

Description: This function returns the current selector:offset address for the specified protected mode interrupt handler.

Parameters: number interrupt number

Return value: addr protected mode selector:offset vector of interrupt

Error code: $0002

Notes: The address returned is a protected mode selector:offset address, not a real mode address. All 256 interrupts are supported by the DPMI host.

See also: Interrupt services,(108), dpmisetpminterrupt (111) dpmisetexceptionhandler (110), dpmigetexceptionhandler (110), dpmigetrminterrupt (109) dpmisetrminterrupt (109), dpmiallocatermcallback (114), dpmisimulaterminterrupt (113) intr (120), realintr (120), pm_addr (97), table (13.2)

13.6.37 dpmi_set_pm_interrupt

Declaration: `function dpmi_set_pm_interrupt(number : Byte; addr : pm_Addr) : Boolean;`

Description: Sets the specified protected mode interrupt vector of the current task.

Parameters: number interrupt number to set addr new protected mode address of interrupt.

Return value: none

Error code: $0002

Notes: The address passed to this function must be a valid selector:offset protected mode address.

See also: Interrupt services,(108), dpmigetpminterrupt (111) dpmisetrminterrupt (109), dpmigetrminterrupt (109), dpmisetexceptionhandler (110) dpmigetexceptionhandler (110), dpmiallocatermcallback (114), dpmisimulaterminterrupt (113) intr (120), realintr (120), pm_addr (97), table (13.2)

13.6.38 Virtual interrupt state functions

Under many implementations of DPMI, the interrupt flag in protected mode will always be set (interrupts enabled). This is because the program is running under a protected mode operating system that does not allow programs to disable physical hardware interrupts. However, the operating system will maintain a 'virtual' interrupt state for protected mode programs. When the program exectues a 'cli' instruction, the programs virtual interrupt state will be disabled, and the program will not receive any hardware interrupts until it executes a 'sti' to reenable interrupts.

13.6.39 dpmi_get_and_disable_virtual_interrupts

Declaration: `function dpmi_get_and_disable_virtual_interrupts(var prevstate : Boolean) : Boolean;`

Description: This function will disable the virtual interrupt state and return the previous state of the virtual interrupt flag.

Parameters: none

Return value: prevstate previous virtual interrupt state

Error code:

Notes: Virtual interrupts are disabled after this call.

See also: Virtual interrupt state functions,(111), dpmigetandenablevirtualinterrupts (112), dpmigetvirtualinterruptstate (112) table (13.2)

13.6.40 dpmi_get_and_enable_virtual_interrupts

Declaration: `function dpmi_get_and_enable_virtual_interrupts(var prevstate : Boolean) : Boolean;`

Description: This function will enable virtual interrupts and return the previous state of the virtual interrupt flag

Parameters: none

Return value: prevstate previous virtual interrupt state

Error code:

Notes: Virtual interrupts are enabled after this call

See also: Virtual interrupt state functions,(111), dpmigetanddisablevirtualinterrupts (112), dpmigetvirtualinterruptstate (112) table (13.2)

13.6.41 dpmi_get_virtual_interrupt_state

Declaration: `function dpmi_get_virtual_interrupt_state(var state : Boolean) : Boolean;`

Description: Returns the current state of the virtual interrupt flag

Parameters: none

Return value: state current virtual interrupt flag state

Error code:

Notes: none

See also: Virtual interrupt state functions,(111), dpmigetandenablevirtualinterrupts (112), dpmigetanddisablevirtualinterrupts (112) table (13.2)

13.6.42 Translation services

These services are provided so that protected mode programs can call real mode software that DPMI does not support directly. The registers structure is passed by the DPMI host to these programs.

All functions automatically provide a locked realmode stack of about 30 words.

If your program needs to perform a series of calls to a real mode API it's sometimes more convenient to use the translation services to call a real mode procedure in your own program. That procedure can then issue the API calls in real mode and then return to protected mode.

There is also a mechanism for protected mode software to gain control from real mode via a real mode callback address. Real mode callbacks can be used to hook to real mode interrupts or to be called in protected mode by a real mode driver. For example, many mouse drivers will call a specified address whenever the mouse is moved. This service allows the callback to be handled by software running in protected mode.

13.6.43 dpmi_simulate_rm_interrupt

Declaration: `function dpmi_simulate_rm_interrupt(number : Byte; var regs : Registers): Boolean;`

Description: This function simulates an interrupt in realmode.

Parameters: number interrupt to call regs register values supplied to interrupt

Return value: regs register values changed by interrupt

Error code: $0002

Notes:

See also: Translation services,(113), intr (120) realintr (120), registers (Type),(registers (97)), dpmicallrmprocedurewithiretframe (114) table (13.2)

13.6.44 dpmi_call_rm_procedure_with_retf_frame

Declaration: `function dpmi_call_rm_procedure_with_retf_frame(var regs : Registers) : Boolean;`

Description: This function calls a real mode procedure. The called procedure must end with a far return when it completes.

Parameters: regs register values supplied to realmode procedure

Return value: regs register values modified by real mode procedure

Error code: $0002

Notes: The cs:ip in the register values structure specifies the address of procedure to call. Note that all given segment registers in this structure need to be real mode segments.

See also: Translation services,(113), table (13.2)

13.6.45 dpmi_call_rm_procedure_with_iret_frame

Declaration: `function dpmi_call_rm_procedure_with_iret_frame(var regs : Registers) : Boolean;`

Description: This function calls a realmode procedure which must return with an iret instruction

Parameters: regs registers supplied to realmode procedure

Return value: regs register values modified by realmode procedure

Error code:

Notes: The cs:ip fields in the register values structure specifies the address of the procedure to call.

See also: Translation services,(113), table (13.2)

13.6.46 dpmi_allocate_rm_callback

Declaration: `function dpmi_allocate_rm_callback(procaddr : pm_Addr; regs : pm_Addr; var realaddr : rm_Addr) : Boolean;`

Description: This function is used to obtain a unique realmode segmentoffset address that will transfer control from realmode to a protected mode procedure.

Parameters: procaddr selector:offset of procedure to call regs selector:offset of real mode call structure

Return value: realaddr segment:offset of real mode call address

Error code:

Notes: Callback procedure parameters at entry: Interrupts disabled

Return from callback procedure: Execute an iret to return

The called procedur is responsible for modifying the realmode CS:IP before returning. If the real mode CS:IP is left unchanged the realmode callback will be executed immediately and your protected mode procedure will be called again. Normally you'll pop a return address off the realmode stack and place it in the real mode CS:IP.

To return values to the realmode caller you must modify the realmode call structure. Remember that all segment values in the real mode call structure will be realmode segments, not selectors.

See also: Translation services,(113), table (13.2)

13.6.47 dpmi_free_rm_callback

Declaration: `function dpmi_free_rm_callback(procaddr : rm_Addr) : Boolean;`

Description: This function frees a real mode callback address that was previously allocated through the dpmi_allocate_rm_callback() function.

Parameters: rm_Addr realmode segment:offset callback address to free

Return value: none

Error code: $0002

Notes: Real mode callbacks are a limited resource. They should be freed when no longer used.

See also: Translation services,(113), table (13.2)

13.6.48 Page locking services

Many DPMI implementations supply virtual memory (as CWSDPMI and Win9x does). In these environments it is necessary to lock any memory that can be touched while executing inside of DOS. This is necessary because it may not be possible for the operating system to demand load a page while DOS is busy. Under all DPMI implementations, applications should lock interrupt code and data. The lock calls will always return success under implementations that ignore these calls.

Although memory ranges are in specified in bytes, the actual unit of memory that is locked will be one or more pages. Page locks are maintained as a count. When the count is decremented to zero, the page is unlocked and can be swapped to disk. This means if a region is locked three times then it must be unlocked three times before the pages will be unlocked.

13.6.49 dpmi_lock_linear_region

Declaration: `function dpmi_lock_linear_region(linaddr : DWord; size : DWord) : Boolean;`

Description: This function locks a specified memory range

Parameters: linaddr starting linear address of memory range to lock size size of region to lock in bytes

Return value: none

Error code:

Notes: If this function fails, no memory was locked. If the specified region overlaps part of a page at the beginning or end, the whole page(s) will be locked.

See also: Page locking services,(115), table (13.2)

13.6.50 dpmi_unlock_linear_region

Declaration: `function dpmi_unlock_linear_region(linaddr : DWord; size : DWord) : Boolean;`

Description: Unlocks a specified linear address memory range that was previously locked by dpmi_lock_linear_region().

Parameters: linaddr starting linear address of memory range size size of region to be locked in bytes

Return value: none

Error code: $0002

Notes: If the function fails, none of the memory will be unlocked. An error will be returned if the memory was not previously locked or if the specified region is invalid. If the specified region overlaps part of a page at the beginning or end, the page(s) will be unlocked. Even if the function succeeds, the memory will remain locked if the lock count is not decremented to zero.

See also: Page locking services,(115), table (13.2)

13.6.51 dpmi_mark_rm_region_as_pageable

Declaration: `function dpmi_mark_rm_region_as_pageable(startaddr : DWord; size : DWord) : Boolean;`

Description: Normally some DPMI implementations lock the DOS real mode memory by default to prevent disk swapping of this memory. If a protected mode program is using DOS memory, it is a good idea to use this function to turn off automatic page locking for regions of memory that are not touched by interrupts

Parameters: startaddr starting linear address of memory to be marked as pageable size size of memory to page in bytes

Return value: none

Error code:

Notes: Do not mark memory not owned by your program as pageable. It is very important to relock any real mode memory using dpmi_relock_rm_region() before terminating the program. Memory that remains unlocked after program exit may cause fatal page faults when other software is accessing the same address space. Note that address space marked as pageable may be locked by using dpmi_lock_linear_region(). This function is only an advisory for the DPMI host to allow memory that doesn't be locked to be paged out. This function just disables any automatic locking of realmode memory performed by the DPMI host. If this function fails then none of the memory will be unlocked. If the specified region overlaps part of a page at the beginning or end these page(s) will not marked as pageable.

See also: Page locking services,(115), table (13.2)

13.6.52 dpmi_relock_rm_region

Declaration: `function dpmi_relock_rm_region(startaddr : DWord; size : DWord) : Boolean;`

Description: This function is used to relock realmode memory previously locked via dpmi_mark_rm_region_as_pageable().

Parameters: startaddr starting linear address of memory to be relocked size size of memory region in bytes

Return value: none

Error code:

Notes: If this function fails, none of the memory specified will be relocked. If the specified region overlaps part of a page at the beginning or end of the region, the page(s) will not be relocked.

See also: Page locking services,(115), table (13.2)

13.6.53 dpmi_get_page_size

Declaration: `function dpmi_get_page_size(var pagesize : DWord) : Boolean;`

Description: Returns the size of a single memory page

Parameters: none

Return value: pagesize size of page in bytes

Error code: none (this function never fails)

Notes: The typical size is 4 kb, but don't make any assumptions on this.

See also: Page locking services,(115), table (13.2)

13.6.54 Demand paging performance tuning services

Some applications will discard memory objects or will not access memory objects for long periods of time. These services can be used to improve performance of demand paging.

Although these functions are only relevant for DPMI implementations that support virtual memory (both CWSDPMI and Win9x do this), other implementations may ignore this (although they always succeed).

Since both of this functions are simply advisory, the operating system may choose to ignore these calls.

13.6.55 dpmi_mark_page_as_demand_paging_candidate

Declaration: `function dpmi_mark_page_as_demand_paging_candidate(startaddr : DWord; size : DWord) : Boolean;`

Description: This function is used to inform the operating system that a range of pages should be placed on top of the page out candidate list. This will force these pages to be swapped to disk even if they were accessed recently. However all memory contents will be preserved.

Parameters: startaddr starting linear address of pages to mark size size of region in bytes to mark as paging candidates

Return value: none

Error code:

Notes: This function does not force the pages to be swapped immediately. Partial pages will not be marked.

See also: Demand paging performance tuning services,(117), table (13.2)

13.6.56 dpmi_discard_page_contents

Declaration: `function dpmi_discard_page_contents(startaddr : DWord; size : DWord) : Boolean;`

Description: This function discards the entire memory contents of a given linear memory range. It is used after a memory object that occupied a given piece of memory has been discarded.

Parameters: startaddr starting linear address of pages to discard size number of bytes to discard

Return value: none

Error code:

Notes: The contents of the memory region will be undefined the next time memory is accessed. All values previously stored in this memory will be lost. Partial pages will not be discarded.

See also: Demand paging performance tuning services,(117), table (13.2)

13.6.57 Miscellaneous services

13.6.58 dpmi_get_version

Declaration: `function dpmi_get_version(var ver : Word; var flags : Word; var processor : Byte; picbase : Word) : Boolean;`

Description: Returns the version of the DPMI services supported.

Parameters: none

Return value: ver the hibyte is the major version number, the lowbyte the minor version number flags refer to DPMI specification processor processor type (2=286..4=486) picbase refer to DPMI specification

Error code: $0002

Notes: none :)

See also: Miscellaneous services (118), table (13.2)

13.6.59 Commonly used combinations of the above

("Time savers") These are functions which do multiple calls to the DPMI unit for often repeated function sequences. This was done in order to make your life easier and the code more understandable.

A list which DPMI functions what function calls can be seen in Table 5.

13.6.60 create_selector

Declaration: `function create_selector(var sel : Word; baseaddr, size : DWord) : Boolean;`

Description: Allocates a single new selector and initializes it to the specified values

Parameters: baseaddr linear 32 bit base address of new selector size new 32 bit segment limit in bytes 1

Return value: none

Error code: See notes

Notes: This function calls several different DPMI functions, the error code returned is the one from the DPMI functions called which caused the error.

See also: Commonly used combinations (118), table (13.5)

13.6.61 change_selector

Declaration: `function change_selector(sel : Word; new_baseaddr, new_size : DWord) : Boolean;`

Description: Changes base address and limit of a single selector.

Parameters: new_baseaddr new linear 32 bit base address of selector new_size new 32 bit segment limit in bytes 1

Return value: none

Error code: See notes

Notes: This function calls several different DPMI functions, the error code returned is the one from the DPMI functions called which caused the error.

See also: Commonly used combinations (118), table (13.5)

13.6.62 free_selector

Declaration: `function free_selector(sel : Word) : Boolean;`

Description: Frees a previously allocated selector

Parameters: sel selector to free

Return value: none

Error code: See dpmi_free_ldt_descriptor()

Notes: See dpmi_free_ldt_descriptor()

See also: Commonly used combinations (118), table (13.5)

13.6.63 get_linear_address

Declaration: `function get_linear_address(sel : Word; offset : DWord) : DWord;`

Description: Returns the 32 bit linear address of a protected mode selector:offset address

Parameters: sel selector offset offset into selector

Return value: 32 bit linear address

Error code: See dpmi_get_segment_base_address()

Notes: See dpmi_get_segment_base_address()

See also: Commonly used combinations (118), table (13.5)

13.6.64 map_physical_memory

Declaration: `function map_physical_memory(physaddr : DWord; size : DWord; var sel : Word) : Boolean;`

Description: Maps a memory area of a physical device to a single descriptor

Parameters: physaddr physical address of device to map size size of memory region to map in bytes 1

Return value: sel selector to memory region

Error code: See notes

Notes: This function uses several different DPMI calls, so the error code returned is the one from the function which caused the error.

See also: Commonly used combinations (118), table (13.5)

13.6.65 intr

Declaration: `function intr(num : Byte; var r : registers) : Boolean;`

Description: Issues a real mode interrupt.

Parameters: num number of interrupt r supplied registers data structure

Return value: r registers data structure with the values changed by the interrupt

Error code: See dpmi_simulate_rm_interrupt()

Notes: See dpmi_simulate_rm_interrupt()

See also: Commonly used combinations (118), table (13.5)

13.6.66 realintr

Declaration: `function realintr(num : Byte; var r : registers) : Boolean;`

Description: Issues a real mod interrupt

Parameters: num number of interrupt r supplied registers data structure

Return value: r register values data structure changed by interrupt

Error code: See dpmi_simulate_rm_interrupt()

Notes: See dpmi_simulate_rm_interrupt()

See also: Commonly used combinations (118), table (13.5)

13.6.67 lock_data

Declaration: `function lock_data(var data; size : DWord) : Boolean;`

Description: Locks a memory range in the

Parameters: data address of data to be locked size size of memory range to be locked

Return value: none

Error code: See dpmi_lock_linear_region()

Notes: See dpmi_lock_linear_region()

See also: Commonly used combinations (118), table (13.5)

13.6.68 lock_code

Declaration: `function lock_code(faddr : pointer; size : DWord) : Boolean;`

Description: Locks a memory range in the

Parameters: faddr starting address to data (typically code) to be locked size size of memory range to be locked

Return value: none

Error code: See dpmi_lock_linear_region()

Notes: See dpmi_lock_linear_region()

See also: Commonly used combinations (118), table (13.5)

13.6.69 unlock_data

Declaration: `function unlock_data(var data; size : DWord) : Boolean;`

Description: Unlocks a memory range in the

Parameters: data address of data to be unlocked size size of memory region to be unlocked

Return value: none

Error code: See dpmi_unlock_linear_region()

Notes: See dpmi_unlock_linear_region()

See also: Commonly used combinations (118), table (13.5)

13.6.70 unlock_code

Declaration: `function unlock_code(faddr : Pointer; size : DWord) : Boolean;`

Description: Unlocks a memory range in the

Parameters: faddr starting address of region to be unlocked size size of memory range in bytes

Return value: none

Error code: See dpmi_unlock_linear_region()

Notes: See dpmi_unlock_linear_region()

See also: Commonly used combinations (118), table (13.5)

13.6.71 Segment registers access

The following functions give FPC programs full read access to the different segment registers.

13.6.72 CSeg

Declaration: `function CSeg :  Word;`

Description: Returns the contents of the

Parameters: none

Return value: current value of

Error code: none (This function never fails)

Notes: It is allowed under FPC (GO32V2) to write to the code segment by default.

See also: Segment register access (121), dseg (122), dsegalias (122) eseg (123), fseg (123), gseg (123), sseg (123),

13.6.73 DSeg

Declaration: `function DSeg :  Word;`

Description: Returns the contents of the

Parameters: none

Return value: current value of

Error code: none (This function never fails)

Notes: rely on this

See also: Segment register access (121), cseg (122), dsegalias (122) eseg (123), fseg (123), gseg (123), sseg (123),

13.6.74 DSegAlias

Declaration: `function DSegAlias :  Word;`

Description: Returns ___djgpp_ds_alias, which is a copy of the

Parameters: none

Return value: value of ___djgpp_ds_alias

Error code: none (This function never fails)

Notes: The only difference between ___djgpp_ds_alias isn't set to zero at a runtime error. This is used by FPC so that the program automatically exits at the next It is good to use this value instead of the standard realmode callbacks, so that runtime errors do not occur while executing in realmode (done automatically).

See also: Segment register access (121), cseg (122), dseg (122), eseg (123) fseg (123), gseg (123), sseg (123),

13.6.75 ESeg

Declaration: `function ESeg :  Word;`

Description: Returns the contents of the

Parameters: none

Return value: current value of

Error code: none (This function never fails)

Notes: rely on this

See also: Segment register access (121), cseg (122), dseg (122), dsegalias (122) fseg (123), gseg (123), sseg (123),

13.6.76 FSeg

Declaration: `function FSeg :  Word;`

Description: Returns the contents of the

Parameters: none

Return value: current value of

Error code: none (This function never fails)

Notes: functions rely on this, so make sure that you restore it after use.

See also: Segment register access (121), cseg (122), dseg (122), dsegalias (122) eseg (123), gseg (123), sseg (123),

13.6.77 GSeg

Declaration: `function GSeg :  Word;`

Description: Returns the contents of the

Parameters: none

Return value: current value of

Error code: none (This function never fails)

Notes: This is the only segment register that can be changed without restoring it.

See also: Segment register access (121), cseg (122), dseg (122), dsegalias (122) eseg (123), fseg (123), sseg (123)

13.6.78 SSeg

Declaration: `function SSeg :  Word;`

Description: Returns the contents of the

Parameters: none

Return value: current value of

Error code: none (This function never fails)

Notes:

See also: Segment register access (121), cseg (122), dseg (122), dsegalias (122) eseg (123), fseg (123), gseg (123)

13.6.79 Port access

Port accesses are done via the following functions using DPMI in FPC

13.6.80 outportb

Declaration: `procedure outportb(port : Word; data : Byte);`

Description: Sends a single byte to the specified port address

Parameters: port port address to send data to data data byte sent

Return value: none

Error code: none (This function never fails)

Notes:

See also: Port access,(124), outportw (124) outportl (124), inportb (125) inportw (125), inportl (125)

13.6.81 outportw

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

Description: Sends a single word to the specified port address

Parameters: port port address to send data to data data word sent

Return value: none

Error code: none (This function never fails)

Notes:

See also: Port access,(124), outportb (124) outportl (124), inportb (125), inportw (125), inportl (125),

13.6.82 outportl

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

Description: Sends a single longint to the specified port address

Parameters: port port address to send data to data data longint sent

Return value: none

Error code: none (This function never fails)

Notes:

See also: Port access,(124), outportb (124) outportw (124), inportb (125) inportw (125), inportl (125)

13.6.83 inportb

Declaration: `function inportb(port: Word) : Byte;`

Description: Reads a single byte from the specified port address

Parameters: port Port address to get data from

Return value: Data byte read

Error code: none (This function never fails)

Notes:

See also: Port access,(124), outportb (124) outportw (124), outportl (124) inportw (125), inportl (125)

13.6.84 inportw

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

Description: Reads a single word from the specified port address

Parameters: port Port address to get data from

Return value: Data word read

Error code: none (This function never fails)

Notes:

See also: Port access,(124), outportb (124) outportw (124), outportl (124) inportb (125), inportl (125)

13.6.85 inportl

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

Description: Reads a single longint from the specified port address

Parameters: port Port address to get data from

Return value: Data Longint read

Error code: none (This function never fails)

Notes:

See also: Port access,(124), outportb (124) outportw (124), outportl (124) inportb (125), inportw (125)

13.6.86 Enable / disable hardware interrupts

Enable or disable hardware interrupts in time critical code.

13.6.87 Enable

Declaration: `procedure Enable;`

Description: Enables hardware interrupts

Parameters: none

Return value: none

Error code: none (This function never fails)

Notes: Issues a sti instruction

See also: Enable/Disable hardware interrupts,(125), disable (126),

13.6.88 Disable

Declaration: `procedure Disable;`

Description: Disables all hardware interrupts

Parameters: none

Return value: none

Error code: none (This function never fails)

Notes: Hardware interrupts shouldn't be turned off for longer periods of time, because several components of a modern PC rely on this.

See also: Enable/Disable hardware interrupts,(125), enable (126),

13.6.89 Transfer buffer access

The transfer buffer is a preallocated DOS memory area, which can be freely used by applications for temporary use, e.g. buffers for realmode interrupts which demand that the buffer is located in realmode memory area (¡1MB boundary).

13.6.90 tb_size

Declaration: `function tb_size : DWord;`

Description: Returns the size of the preallocated transfer buffer.

Parameters: none

Return value: Size in bytes of the buffer

Error code: none (This function never fails)

Notes: Don't make any assumptions about the size of this buffer, although it is typically 16 kb.

See also: Transfer buffer,(126), memory management,(105), tbsize (126),

13.6.91 tb_address

Declaration: `function tb_address : DWord;`

Description: Returns the linear offset of the transfer buffer from the

Parameters: none

Return value: Offset of buffer from the beginning of the

Error code: none (This function never fails)

Notes: The fseg() function can be used to determine the full 48 bit protected mode selector:offset address for copying purposes. The real mode segment:offset address of this block is (tb_address() shr 4):(tb_address() and $F) as usual..

See also: Transfer buffer,(126), memory management,(105), tbaddress (127),

13.6.92 "Near pointer" handling

Accessing memory via full 48 bit pointers (selector:offset) is sometimes very annoying, especially when you need to have access to devices that don't reside in the video cards. Especially if speed is a matter, the repeated reloading of segment registers slows down code execution dramatically since this takes several cycles on modern cpu's. The solution is extending a segment's limit to 4 GB and then accessing the memory via a single 32 bit "nearpointer" (also called FLAT pointers).

This is what the following functions do: They extend the limit of the Since the limit doesn't affect system functions and procedures, everything works fine as long as FPC does not try to grow the heap. FPC does this in a way that the a 'Not enough memory' error, because it notices that the segment limit is at the maximum and it can't increase it any further.

So you have several options when designing programs:

know how much memory you need . Since FPC's heap won't grow then anymore it's no problem (with the Chxxxxxx compiler option)

every time you allocate memory in any way, restore the old segment registers (by using dpmi_disable_nearptr()). After allocating memory as normal, and reenabling nearpointer mapping you need to restore the pointers to external devices you got dpmi_nearptr_address_mapping() (typically you'll only need one, a pointer to the VGA)

the best way is to allocate all needed memory before you enable near pointer mapping

Use nearpointer mapping carefully ! This may destroy everything on your computer, since your program can write everywhere to memory space then ;) (It's like it was in realmode times then, only with the difference that you have up to 4 GB of memory available)

13.6.93 dpmi_enable_nearptr

Declaration: `function dpmi_enable_nearptr : Boolean;`

Description: Enables "nearpointer" mapping by setting 32 bit limit of to $FFFFFFFF

Parameters: none

Return value: none

Error code: $F000

Notes: Near pointer mapping can be disabled at any time by issuing a dpmi_disable_nearptr() call.

See also: "Near pointer" handling,(127), dpmidisablenearptr (128) dpminearptrenabled (128), dpminearptraddressmapping (128),

13.6.94 dpmi_disable_nearptr

Declaration: `function dpmi_disable_nearptr : Boolean;`

Description: Disables "nearpointer" mapping by restoring the old values of and

Parameters: none

Return value: none

Error code: $F001

Notes: "Near pointer" mapping can be enabled and disabled at any time, but once you disabled near pointers, you'll need to restore pointers which where gained via dpmi_nearptr_address_mapping() again since the base segment address may have changed. All other FPC functions and procedures work as usual.

See also: "Near pointer" handling,(127), dpmienablenearptr (127) dpminearptrenabled (128), dpminearptraddressmapping (128)

13.6.95 dpmi_nearptr_enabled

Declaration: `function dpmi_nearptr_enabled(var enabled : Boolean) : Boolean;`

Description: Returns the current status of near pointer mapping

Parameters: none

Return value: enabled true if "near pointers" are enabled, false if not

Error code: none (This function never fails)

Notes: none

See also: "Near pointer" handling,(127), dpmienablenearptr (127) dpmidisablenearptr (128), dpminearptraddressmapping (128),

13.6.96 dpmi_nearptr_address_mapping

Declaration: `function dpmi_nearptr_address_mapping(physaddr, size : DWord; var nearptr : Pointer) : Boolean;`

Description: Returns a 32 bit "near pointer" to a physical memory area (e.g. VGA)

Parameters: physaddr the physical address of the device size the memory size of the physical device to be mapped 1

Return value: nearptr the 32 bit "near pointer" to the physical memory area

Error code: $F002

Notes: Fails if not in near pointer mode. This function uses several different other DPMI functions, if they fail, they return their own error code.

See also: "Near pointer" handling,(127), dpmienablenearptr (127) dpmidisablenearptr (128), dpminearptrenabled (128),

13.7 Appendix A : index

table1

(Replaced by Tex generated index at start of document)

13.8 Appendix B : Error codes

Table 13.2: Error codes returned by DPMI functions

DPMI function	Error codes returned	Notes
function dpmi_get_cpu_mode (99)	(none)	errcx1
function dpmi_allocate_ldt_descriptors (100)	$0000	
function dpmi_free_ldt_descriptor (100)	$0001	
function dpmi_segment_to_descriptor (101)	$0002	
function dpmi_get_next_selector_increment_value (101)	($0003)	errcx1
function dpmi_get_segment_base_address (101)	$0006	
function dpmi_set_segment_base_address (102)	$0007	
function dpmi_get_segment_limit (102)	See notes	errcx2
function dpmi_set_segment_limit (102)	$0008	
function dpmi_get_descriptor_access_rights (103)	See notes	errcx2
function dpmi_set_descriptor_access_rights (103)	$0009	
function dpmi_create_code_segment_alias_descriptor (103)	$000A	
function dpmi_get_descriptor (104)	$000B	
function dpmi_set_descriptor (104)	$000C	
function dpmi_allocate_specific_descriptor (104)	$000D	
function dpmi_get_free_memory_information (105)	($0500)	errcx1
function dpmi_allocate_memory_block (105)	$0501	
function dpmi_free_memory_block (106)	$0502	
function dpmi_resize_memory_block (106)	$0503	
function dpmi_physical_address_mapping (106)	$0800	
function dpmi_allocate_dos_memory_block (107)	$0100	
function dpmi_free_dos_memory_block (108)	$0101	
function dpmi_resize_dos_memory_block (108)	$0102	
function dpmi_get_rm_interrupt (109)	($0200)	errcx1
function dpmi_set_rm_interrupt (109)	$0201	
function dpmi_get_exception_handler (110)	$0202	
function dpmi_set_exception_handler (110)	$0203	
function dpmi_get_pm_interrupt (111)	$0204	
function dpmi_set_pm_interrupt (111)	$0205	
function dpmi_get_and_disable_virtual_interrupts (112)	($0900)	errcx1
function dpmi_get_and_enable_virtual_interrupts (112)	($0901)	errcx1
function dpmi_get_virtual_interrupt_state (112)	($0902)	errcx1
function dpmi_simulate_rm_interrupt (113)	$0300	
function dpmi_call_rm_procedure_with_retf_frame (113)	$0301	
function dpmi_call_rm_procedure_with_iret_frame (114)	$0302	
function dpmi_allocate_rm_callback (114)	$0303	
function dpmi_free_rm_callback (114)	$0304	

Table 13.2: Error codes returned by DPMI functions

DPMI function	Error codes returned	Notes
function dpmi_get_version (118)	$0400	errcx1
function dpmi_lock_linear_region (115)	$0600	
function dpmi_unlock_linear_region (115)	$0601	
function dpmi_mark_rm_region_as_pageable (116)	$0602	
function dpmi_relock_rm_region (116)	$0603	
function dpmi_get_page_size (116)	$0604	
function dpmi_mark_page_as_demand_paging_candidate (117)	$0702	
function dpmi_discard_page_contents (117)	$0703	
function create_selector (118)	See notes	errcx3
function change_selector (119)	See notes	errcx3
function free_selector (119)	See notes	errcx3
function get_linear_address (119)	See notes	errcx3
function map_physical_memory (119)	See notes	errcx3
function intr (120)	See notes	errcx3
function realintr (120)	See notes	errcx3
function lock_data (120)	See notes	errcx3
function lock_code (121)	See notes	errcx3
function unlock_data (121)	See notes	errcx3
function unlock_code (121)	See notes	errcx3
function dpmi_enable_nearptr (127)	$F000	errcx4
function dpmi_disable_nearptr (128)	$F001	
function dpmi_nearptr_enabled (128)	(none)	errcx1
function dpmi_nearptr_address_mapping (128)	$F002	

Notes:
1 This function always succeeds, so you'll never encounter a error here
2 Check the boolean result for success/fail
3 This is a combination of several subsequent DPMI calls. The error code is the one returned by the DPMI call failed
4 May occur both as warning or as fatal error

13.9 Appendix C : Go32 and DPMI comparison

Table 13.3: Go32 and DPMI equivalents

GO32 term	Equivalent DPMI term	Notes
Constants		
carryflag	flag constants 97	conv1
parityflag	flag constants 97	conv1
auxcarryflag	flag constants 97	conv1
zeroflag	flag constants 97	conv1
signflag	flag constants 97	conv1
trapflag	flag constants 97	conv1
interruptflag	flag constants 97	conv1
directionflag	flag constants 97	conv1
overflowflag	flag constants 97	conv1
Types		
tmeminfo	MemInfoBuf (98)	
tseginfo	pm_adddr (97), rm_adddr (97)	
trealregs	registers (97)	
registers	registers (97)	
Variables		

Table 13.3: Go32 and DPMI equivalents

GO32 term	Equivalent DPMI term	Notes
dosmemselector	fseg (123)	
int31error	dpmi_error (98), dpmi_set_error_handler (99)	conv2
Functions & Procedures		
allocate_ldt_descriptors()	dpmi_allocate_ldt_descriptors (100)	
	create_selector (118)	
free_ldt_descriptor()	dpmi_free_ldt_descriptor (100)	
	free_selector (119)	
segment_to_descriptor()	dpmi_segment_to_descriptor (101)	
get_next_selector_increment_value()	dpmi_get_next_selector_increment_value (101)	
get_segment_base_address()	dpmi_get_segment_base_address (101)	
set_segment_base_address()	dpmi_set_segment_base_address (102)	
	create_selector (118)	
set_segment_limit()	dpmi_set_segment_limit (102)	
set_descriptor_access_rights()	dpmi_get_descriptor_access_rights (103)	
create_code_segment_alias_descriptor()	dpmi_create_code_segment_alias_descriptor (103)	
get_linear_addr()	dpmi_physical_address_mapping (106)	
	map_physical_memory (119)	
	get_linear_address (119)	
get_segment_limit()	dpmi_get_segment_limit (102)	
get_descriptor_access_right()	dpmi_get_descriptor_access_rights (103)	
get_page_size()	dpmi_get_page_size (116)	
map_device_in_memory_block()	notes	conv3
realintr()	dpmi_simulate_rm_interrupt (113), intr (120), realintr (120)	
global_dos_alloc()	dpmi_allocate_dos_memory_block (107)	
global_dos_free()	dpmi_free_dos_memory_block (108)	
seg_fillchar()		notes
seg_fillword()		notes
get_meminfo()	dpmi_get_free_memory_information (105)	
get_pm_interrupt()	dpmi_get_pm_interrupt (111)	
set_pm_interrupt()	dpmi_set_pm_interrupt (111)	
get_rm_interrupt()	dpmi_get_rm_interrupt (109)	
set_rm_interrupt()	dpmi_set_rm_interrupt (109)	
get_exception_handler()	dpmi_get_exception_handler (110)	
set_exception_handler()	dpmi_set_exception_handler (110)	
get_pm_exception_handler()	notes	conv3, conv4
set_pm_exception_handler()	notes	conv3, conv4
free_rm_callback()	dpmi_free_rm_callback (114)	
get_rm_callback()	dpmi_allocate_rm_callback (114)	
get_cs()	cseg (122)	conv1
get_ds()	dseg (122), dsegalias (122)	conv1
get_ss()	sseg (123)	conv1
allocate_memory_block()	dpmi_allocate_memory_block (105)	conv5, conv6
free_memory_block()	dpmi_free_memory_block (106)	conv6
request_linear_region()	notes	conv3, conv4
lock_linear_region()	dpmi_lock_linear_region (115)	
lock_data()	lock_data (120)	
lock_code()	lock_code (121)	
unlock_linear_region()	dpmi_unlock_linear_region (115)	
unlock_data()	unlock_data (121)	
unlock_code()	unlock_code (121)	
disable()	disable (126)	
enable()	enable (126)	
inportb()	inportb (125)	
inportw()	inportw (125)	

Table 13.3: Go32 and DPMI equivalents

GO32 term	Equivalent DPMI term	Notes
inportl()	inportl (125)	
outportb()	outportb (124)	
outportw()	outportw (124)	
outportl()	outporl (124)	
get_run_mode()	notes	conv7
transfer_buffer()	tb_address (127)	
tb_segment()	tb_address (127)	
tb_offset()	tb_address (127)	
tb_size()	tb_size (126)	
copytodos()	notes	conv8
copyfromdos()	notes	conv8
dosmemput()	notes	conv8
dosmemget()	notes	conv8
dosmemmove()	seg_memcpy()	conv9
dosmemfillchar()	seg_memsetb()	conv9
dosmemfillword()	seg_memsetw()	conv9

Notes:

- 1 Actually more than these are supplied by DPMI
- 2 The dpmi_error vrriable (dpmi_error (98)) is updated by the errorhandler supplied by dpmi_set_error_handler (99) by default.
- 3 DPMI 1.0 function, does not work at all.
- 4 Works only with CWSDPMI under real DOS (not Windows9x DOS-Box) in FPC
- 5 Buggy implementation in GO32
- 6 DPMI allows resizing of these blocks via dpmi_resize_memory_block (106) too
- 7 Obsolete, because DPMI only supports GO32V2 model
- 8 Use equivalent seg_memcpy() or seg_memsetX() commands supplied by MEMORY.
- 9 See the documentation for the MEMORY unit (memory.htm included in this package)

13.10 Appendix D : Go32 and DPMI comparison

Btw, this table shows only a direct translation of these GO32 calls. It's often much better and easier to use other functions or "time-saver" functions. Seettable5

Table 13.4: Go32DPMI syntax example

GO32 syntax example	DPMI syntax example	Not
selector := allocate_ldt_descriptors(2)	dpmi_allocate_ldt_descriptors (100)(2,selector)	nta
free_ldt_descriptor(selector)	dpmi_free_ldt_descriptor (100)(selector)	nta
selector := segment_to_descriptor($A000)	dpmi_segment_to_descriptor (101)($A000,selector)	
selincr := get_next_selector_increment_value()	dpmi_get_next_selector_increment_value (101)(selincr)	
base_address := get_segment_base_address(selector)	dpmi_get_segment_base_address (101) (selector,base_address)	
set_segment_base_address(selector, new_address)	dpmi_set_segment_base_address (102)(selector,new_address)	nta
set_segment_limit(selector, new_limit)	dpmi_set_segment_limit (102)(selector,limit)	
set_descriptor_access_rights(selector, new_rights)	dpmi_get_descriptor_access_rights (103)(selector,new_rights)	
datasel := create_code_segment_alias_descriptor(code_sel)	dpmi_create_code_segment_alias_descriptor (103) (code_sel,data_sel)	
linear_addr := get_linear_addr(phys_addr, size)	dpmi_physical_address_mapping (106)(phys_addr,size, linear_addr)	nsm
limit := get_segment_limit(selector)	dpmi_get_segment_limit (102)(selector,limit)	

Table 13.4: Go32DPMI syntax example

GO32 syntax example	DPMI syntax example	Not
rights := get_descriptor_access_right(selector)	dpmi_get_descriptor_access_rights (103)(selector,rights)	
page_size := get_page_size()	dpmi_get_page_size (116)(page_size)	
map_device_in_memory_block()	See notes	nsm
realintr(reg_num, reg_buffer)	dpmi_simulate_rm_interrupt (113)(reg_num,reg_buffer)	nsm
result := global_dos_alloc(size);		
realseg := word(result);		
dos_selector := word(result shr 16);	dpmi_allocate_dos_memory_block (107)(size,realseg, dos_selector)	
global_dos_free(dos_selector)	dpmi_free_dos_memory_block (108)(dos_selector)	
seg_fillchar(selector, offset, size, char(byte_value))	seg_memsetb(selector, offset, size, byte_value)	nsm
seg_fillword(selector, offset, size, word_value)	seg_memsetw(selector, offset, size, word_value)	nsm
get_meminfo(buffer)	dpmi_get_free_memory_information (105)(buffer)	
get_pm_interrupt(number, tseginfo_buffer)	dpmi_get_pm_interrupt (111)(number,pm_Addr(buffer))	nsm
set_pm_interrupt(number, tseginfo_buffer)	dpmi_set_pm_interrupt (111)(number,pm_Addr(buffer))	nsm
get_rm_interrupt(number, tseginfo_buffer)	dpmi_get_rm_interrupt (109)(number,rm_Addr(buffer))	nsm
set_rm_interrupt(number, tseginfo(buffer))	dpmi_set_rm_interrupt (109)(number,rm_Addr(buffer))	nsm
get_exception_handler(number, tseginfo(buffer))	dpmi_get_exception_handler (110)(number,pm_Addr(buffer))	nsm
set_exception_handler(number, tseginfo(buffer))	dpmi_set_exception_handler (110)(number,pm_Addr(buffer))	nsm
get_pm_exception_handler()	See notes	nsm
set_pm_exception_handler()	See notes	nsm
free_rm_callback(tseginfo(intaddr))	dpmi_free_rm_callback (114)(rm_Addr(intaddr))	
get_rm_callback(@func_addr,	dpmi_allocate_rm_callback (114) (pm_Addr(func_Addr),	
registers, tseginfo(rmcb))	pm_Addr(registers), rm_Addr(rmcb))	nsm
cs := get_cs()	cs := cseg (122)	nsm
ds := get_ds()	ds := dseg (122)	nsm
ss := get_ss()	ss := sseg (123)	nsm
linaddr := allocate_memory_block(size)	dpmi_allocate_memory_block (105)(size,handle, linaddr))	nsm
free_memory_block(handle)	dpmi_free_memory_block (106)(handle)	nsm
function request_linear_region()	See notes	nsm
lock_linear_region(linear_addr, size)	dpmi_lock_linear_region (115)(linear_addr,size)	nsm
lock_data(buffer, size)	lock_data (120)(buffer, size)	nsm
lock_code(func_addr, size)	lock_code (121)(func_addr, size)	nsm
unlock_linear_region(linear_addr, size)	dpmi_unlock_linear_region (115)(linear_addr,size)	nsm
unlock_data(buffer, size)	unlock_data (121)(buffer, size)	nsm
unlock_code(func_addr, size)	unlock_code (121)(func_addr, size)	nsm
disable()	disable (126)	nsm
enable()	enable (126)	nsm
data := inportb(port)	data := inportb (125)(port)	nsm
data := inportw(port)	data := inportw (125)(port)	nsm
data := inportl(port)	data := inportl (125)(port)	nsm
outportb(port, data)	outportb (124)(port, data)	nsm
outportw(port, data)	outportw (124)(port, data)	nsm
outportl(port, data)	outportl (124)(port, data)	nsm
get_run_mode()	See notes	nsm
address := transfer_buffer()	address := tbaddress (127)	nsm
seg := tb_segment()	seg := tb_address (127) shr 4	
ofs := tb_offset()	ofs := tb_address (127) and$0F	
size := tb_size()	size := tb_size (126)	nsm
copytodos(buffer, size)	seg_memcpy(dseg, dword(buffer), fseg, tb_address(), size)	nsm
copyfromdos(buffer, size)	seg_memcpy(fseg, tb_address(), dseg, dword(@address), size))	nsm
dosmemput(seg, ofs, buffer, size)	seg_memcpy(dseg, dword(@buffer), fseg, seg shl 4 + ofs, size)	nsm
dosmemget(seg, ofs, buffer, size)	seg_memcpy(fseg, seg shl 4 + ofs, dseg, dword(@buffer), size)	nsm
dosmemmove(srcseg, srcofs, dstseg, dstofs, size)	seg_memcpy(fseg, srcseg shl 4 + ofs, fseg,	
	dstseg shl4 + dstofs, size)	nsm
dosmemfillchar(seg, ofs, size, char(value))	seg_memsetb(fseg, seg shl 4 + ofs, size, byte(value))	nsm
dosmemfillword(seg, ofs, size, value)	seg_memsetw(fseg, seg shl 4 + ofs, size, value)	nsm

Table 13.4: Go32DPMI syntax example

GO32 syntax example	DPMI syntax example	Not
seg_move(srcsel, srcofs, dstsel, dstofs, size)	seg_memcpy(srcsel, srcofs, dstsel, dstofs, size)	nsn
seg_fillchar(sel, ofs, size, char(value))	seg_memsetb(sel, ofs, size, char(value))	nsn
seg_fillword(sel, ofs, size, value)	seg_memsetw(sel, ofs, size, value)	nsn

Notes:

- 1 You could use create_selector() too
- 2 free_selector() could be used too
- 3 Better use map_physical memory for short
- 4 This DPMI 1.0 function won”t work on a DPMI 0.9 host (as CWSDPMI and Win9x DOS box is), so there is no equivalent in DPMI
- 5 Use realintr(), or even easier intr() which does the same
- 6 Same syntax
- 7 By using MEMORY which is automatically included within DPMI via a uses clause
- 8 'Typecasts' indicate that a different type of a passed buffer is expected in DPMI
- 9 dseg_alias() could be used too
- 10 The GO32 function is missing an important parameter (handle)
- 11 Since you don't get a handle for such a GO32 memory block, this function is useless there
- 12 Obsolete since DPMI only works in GO32V2 mode

13.11 Appendix E : ”Time saver” procedures and their equivalent DPMI and GO32 function calls

Table 13.5: ”Tim
GO32 function ca

create_selector (118)(selector,baseaddr, size)	dpmi_allocate_ldt_descriptors (100)(1,selector)
	and dpmi_set_segment_base_address (102)(selector,baseaddr)
	and dpmi_set_segment_limit (102)(selector,size)
change_selector (119)(selector,new_baseaddr,new_size)	dpmi_set_segment_base_address (102)(selector,new_baseaddr)
	and dpmi_set_segment_limit (102)(selector,new_size)
free_selector (119)(selector)	dpmi_free_ldt_descriptors (100)(selector)
get_linear_address (119)(selector,offset)	dpmi_get_segment_base_address (101)(selector,baseaddr) adding offs
map_physical_memory (119)(phys_addr,limit, selector)	dpmi_allocate_ldt_descriptors (100)(1,selector) and dpmi_physical_ad
intr (120)(number, registers)	dpmi_simulate_rm_interrupt (113)(number,registers)
realintr (120)(number, registers)	dpmi_simulate_rm_interrupt (113)(number,registers)
lock_data (120)(buffer, size)	dpmi_lock_linear_region (115)(get_linear_address (119)(dseg,dword(@
lock_code (121)(func_addr, size)	dpmi_lock_linear_region (115)(get_linear_address (119)(cseg,dword(fu
unlock_data (121)(buffer, size)	dpmi_unlock_linear_region (115)(get_linear_address (119)(dseg,dword
unlock_code (121)(buffer, size)	dpmi_unlock_linear_region (115)(get_linear_address (119)(cseg,dword

Index