TEGL Windows Toolkit II
Release 1.10

Programmer's Reference Guide
for TURBO PASCAL 5.0/5.5
and QUICK PASCAL 1.0

TEGL Systems Corporation
Suite 780, 789 West Pender Street
Vancouver, British Columbia
Canada V6C 1H2

TEGL Windows Toolkit II

LICENSE AGREEMENT

TEGL software products are protected under both Canada copyright law and international treaty provisions.

You have the non-exclusive right to use the enclosed software under the following terms and conditions.

You may use this software on a single machine, for both personal and business use; and may make copies of the software solely for backup purposes. Other than this you agree to use this software "like a book", meaning the software may be used by any number of people and may be moved from one computer to another so long as there is no possibility of it being used by more than one person at one time.

Programs that you write and compile using the TEGL Windows Toolkit may be used, given away, or sold without additional license or fees as long as all copies of such programs bear a copyright notice. By "copyright notice" we mean either your own copyright notice or if you prefer, the following statement, "Created using TEGL Windows Toolkit, copyright (C) 1989, 1990, TEGL Systems Corporation. All rights reserved".

Included on the TEGL Windows Toolkit diskettes are a number of support files that contain encoded hardware and font information used by the standard graphic unit. These files are proprietary to TEGL. You may use these files with the programs you create with the TEGL Windows Toolkit for your own personal or business use. To the extent the programs you write and compile using the TEGL Windows Toolkit make use of these support files, you may distribute in combination with such programs, provided you do not use, give away, or sell these support files separately, and all copies of your programs bear a copyright notice.

The Complete Games Toolkit diskettes provide a demonstration on how to use the various features of the TEGL Windows Toolkit. They are intended for educational purposes only. TEGL grants you the right to edit or modify these game programs for your own use but you may not give away, sell, repackage, loan, or redistribute them as part of any program, in executable object or source code form. You may, however, incorporate miscellaneous sample program routines into your programs, as long as the resulting programs do no substantially duplicate all or part of a game program in appearance or functionality and all copies of all such programs bear a copyright notice.

Limited Warranty:

With respect to the physical diskette and physical documentation enclosed herein TEGL warrants same to be free of defects and materials and workmanship for a period of one year from the date of purchase.

TEGL will replace defective Software or documentation upon notice within the warranty period of defects. Remedy for breach of this warranty shall be limited to replacement and shall not encompass any other damages, including, without limitation, loss or business profits, business interruption, pecuniary loss, and special incidental, consequential, or other similar claims. This limited warranty is void if failure of the Software has resulted from accident, abuse, or misapplication. Any replacement Software will be warranted for the remainder of the original warrantly period.

TEGL specifically disclaims all other warranties, express, implied, or statutory, including but not limited to implied warranties of merchantability and fitness for a particular purpose with respect to the Software and documentation. In no event shall TEGL be liable for any loss of business profit or any other commercial damage including but not limited to special, incidental, consequential, or other damages.

TABLE OF CONTENTS

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

SPECIAL NOTE for documentation on disk

You have received Version II of the TEGL Windows Toolkit. The documentation that you are reading is supplied on disk. We will have a printed manual in the near future and it will be somewhat different that what you are looking at now.

Because we wanted everyone to be able to read this manual and be able to print it out we have not embedded any special control characters in it with the exception of formfeeds at page breaks.

In this manual you will notice that at times there are references to things like ctrlkey or keydown or something discriptive but somewhat odd. Please, be imaginative, these will be icons when the manual is printed.

Acknowledgements

In this manual references are made to several products

IBM is a registered trademarks of International Business Machines Inc.

MS-DOS and Windows are registered trademarks of Microsoft Inc.

MacIntosh is a registered trademark of Apple Computer Inc.

Turbo Pascal is a registered trademake of Borland International.

Chapter 1 - Introduction

INTRODUCTION

Welcome to the world of the Graphic User Interface (GUI) in a DOS environment. This book, The programmers reference guide to TEGL Windows Toolkit II, will provide you with the basics (and more) for getting started with using the TEGL Windows Toolkit.

TEGL Windows is a comprehensive GUI toolkit for the simplest to the most complex system programming projects. In order to exploit all the advantages of this toolkit, we encourage you to experiment and to try the examples as listed in this manual.

Programming with TEGL Windows Toolkit

TEGL Windows Toolkit provides the framework to make programs easy to use. If you are new to programming you will find the Toolkit fun and fast to use. Experienced programmers will find their development time reduced by using the Toolkit.

TEGL Windows Toolkit provides graphics that can communicate information more effectively than text. For example, the graphical image of a folder suggests that it contains documents, drawings, and even other folders.

TEGL Windows Toolkit also Provides functions that can build interactive applications. Interactive means a type of user interface where a significant portion of the design and development effort goes into making the program user friendly.

TEGL Windows Toolkit is based on event handling. Events are such things as a key being pressed on the keyboard; a timer signaling that some amount of time has elapsed; a message indicating that the user has selected a particular item from the menu or has selected an icon. A particularly useful capability of this is that while the TEGLSupervisor is waiting for one of these events to occur, you can set the timer to signal a background task such as an internal print spooler. This limited multitasking capability makes it easier to build very interactive programs.

TEGL Windows Toolkit supports only a single application running at any given time. The necessary code is linked into the final application code. TEGL Windows Toolkit uses little RAM, requiring only 50k of the executable program when all features are used.

The Components of TEGL Windows Toolkit

Now that you have a rough idea of what the TEGL Windows Toolkit is, let's explore the components in more detail. The purpose of this section is to give you an overall understanding of how to use the toolkit in your program.

Chapter 1 - Introduction

TEGL Windows Toolkit is subdivided into a set of libraries: multitasking kernel; windowing screen manager; mouse, keyboard and timer handler; a virtual heap manager; drop down and pop-up menu events; and an animation unit.

TEGL Windows Toolkit provides a GUI to a computer running under DOS. This interface is used in a number of entertainment products produced by TEGL Systems Corporation (TSC). As TSC designed and built the entertainment products, TEGL was created to build a set of software routines that were needed by the games. TSC gathered these routines into modules, each categorized by their overall function. For instance, all the routines that manipulate windows form the TEGLUnit. Similarly, all the drop-down menus and menu bars form TEGLMenu.

TEGL Windows Toolkit comprises the tools that were developed in writing the first TSC applications. These tools are now available for developing any application.

The modules are categorized by the kind of functions they deliver; TEGLintr handles the mouse, keyboard and timer interrupts; TEGLMenu provides drop down menus and menu bars; Animate provides icon animation; and VIRTMem provides the virtual heap for almost unlimited windowing ability.

TEGLUnit provides a high level integration between window frames, mouse click areas, keyboard handler, timer interrupts, virtual memory, and multitasking kernel.

What's On your disks

The distribution disks that come with this manual include the complete library of routines used by some of the game products produced by TEGL Systems Corporation.

For your reference, here's a summary of most of the files on disks:

README
: This file contains any last-minute notes and corrections, type README at the system prompt to view the file. You may print this file on your printer for future reference once you review the material.

TEGLUNIT.PAS
: This is the window manager that provides the graphical interface support for the other units. This module provides the event supervisor and the frame/stack coordinator.

TEGLMENU.PAS
: This unit provides the drop down menu interface.

TEGLGRPH.PAS
: This unit provides shadow boxes, shadow texts, exploding and imploding boxes, pop-down/pop-up icon buttons, etc..

TEGLICON.PAS

A library of standard icons; key OK, key CANCEL, key NEXT, key LAST.

TEGLINTR.PAS, TEGLINTR.ASM

Integration of keyboard and mouse handler. This unit provides the standard mouse routines which integrates the keyboard cursor keys and the mouse to provide a seamless dual control of the mouse cursor; with or without a mouse driver.

FASTGRPH.PAS *.ASM

Fast assembly language graphics routines. This is the core of the graphical routines that provide the foundation for pop-down menus and movable windows. This unit includes functions that interfaces with the FASTGRPH and the ANIMATE unit, to allow the recognition of video paging.

TEGLFONT.PAS FNT*.pas

Crisp proportional Bit-Mapped screen fonts, ranging from 6 to 24 pixels in height.

VIRTMEM.PAS

Virtual memory handler that interfaces with TEGLUNIT. This unit automatically pages out images from memory to EMS, hard disk, or floppy (depending on availability), when memory is at a premium. Also implements the far heap for allocating data structures larger than 64K.

SELECTFL.PAS

A standard event unit that may be used by any application program to provide a dialogue window in selecting file names from a list of file on disk.

SENSEMS.PAS

A standard event unit that provides a dialogue window that allows a user to adjust the sensitivity of a mouse.

SOUNDUNT.PAS

A standard event unit that allows a user to adjust the duration and the sound output of a tone.

SWITCHES.INC

Conditional compilaton directive are contained in this file.

ANIMATE.PAS

A unit that allows icons to be animated.

TEGL.PAS

A demonstration program that uses many of the features of the TEGL Windows Toolkit.

FONTTEST.PAS

A demonstration event unit that displays all available fonts in movable windows. Used in tegl.pas (sample program).

DEBUGUNT.PAS

A demonstration event unit that displays general information regarding windows and the number of times the mouse button has been pressed.

SAM*.PAS

Chapter 1 - Introduction

Some of the sample programs in this guide are provided in ready-to-compile form.

EXECSWAP.PAS

A utility unit that swaps a pascal program from memory to enable another program to execute. This unit makes it practical to execute a DOS shell with programs that are using all of memory.

Installing TEGL Windows on your system

The complete TEGL Toolkit is approximately 3 megabytes of source code when expanded. Therefore, a hard disk is required for the installation.

At the DOS prompt, type INSTALL, and follow the instructions.

Development System Requirements

You must have 640k RAM, a hard disk drive, and an EGA/VGA (256K), CGA, or Hercules graphics card and appropriate monitor on an IBM PC or compatible computer. In our development, we've used an IBM PC AT with 2.5MB RAM, 72MB hard disk, and a paradise VGA 256k card with a NEC/MultiSync 3D. We've also tested all our examples on an IBM PC XT with 640k RAM, a 20MB hard disk, and a ATI VIP VGA graphics adapter card with an IBM 8513 VGA color monitor.

Compiling with Turbo Pascal

TEGL Windows Toolkit requires Borland's Turbo Pascal Version 5.0, as a minimum, to compile the units. The Animate unit requires the object oriented programming facilities provided by Version 5.5.

Refer to the Turbo Pascal Reference Manual for including and using units within programs, as well as setting up the environment for referencing the units.

Compiling with Quick Pascal

TEGL Windows works with Microsoft Quick Pascal Version 1.0.

Quick Pascal's integrated environment cannot be used to compile applications using the Toolkit. It runs out of memory. The command-line compiler must be used.

If you intend to use the MSGraph unit then you must define the symbol Quick in the file switches.inc. See the appendix Conditional Compilation for furthur information.

How to use this Reference Manual

This manual is organized in a presentation manner to lead you through the concepts of the TEGL Windows Toolkit II.

Each Procedure and Function is shown seperately with its name, parameter

list, the unit it is declared in, and other references. For a start here is the main entry point into the TEGL Windows Toolkit II.

TEGLSupervisor Procedure TEGLUNIT

Function

Main entry point.

Declaration

TEGLSupervisor;

Remarks

This should be the last statement in your main program block.

Example

```
  BEGIN
    { -- all the setup code for menus etc. goes first }
    TEGLSupervisor;
  END.
```

Program Framework

Most of the examples presented throughout this manual will require the following minimal skeletal Pascal framework before the example code will compile and execute. A few of the examples given are complete programs.

```
{ samshell.pas }
{$F+}  { -- far code model is required for any functions that }
       { -- are to be used as Event Handlers }

Uses
    dos,
    graph,

    virtmem,
    fastgrph,
    TEGLIntr,
    TEGLICON,
    TEGLGRPH,
```

```
     TEGLUnit,
     TEGLMenu,
     TEGLEasy
     SenseMs,
     DebugUnt;


{ -- insert variables here }


{ -- insert procedures and functions here }



BEGIN
   EasyTEGL;

   { -- insert the example code here }
   { -- press Ctrl-Break to exit program }


   { -- control is then passed to the supervisor }

   TEGLSupervisor;
END.
```

Once control has been turned over to the supervisor then the only way to exit a program is by a menu selection or icon that halts the program. Most of the example programs don't have this so you must press Ctrl-Break to exit. When Ctrl-Break is pressed then program control is turned over to an Event Handler. In the case of the sample programs control is passed to Quit in TEGLEasy.

An Event Handler, as covered in Chapter 4, is usually attached to an icon, menu selection, or in this case the Ctrl-Break handler. The Ctrl-Break handler, when attached to an exit event, allows the program to exit gracefully by pressing ctrlkeyscrlock which is the break key on most keyboards.

Chapter 2 provides a foundation to using the TEGL Windows Toolkit by using a few program examples. Chapter 3 shows you how to create an icon using the icon editor, and how to integrate and use the icons in your program. Chapter 4 is heart of the windowing system, which uses most of the other functions provided by the other units. In Chapter 5 we delve further into how the TEGLMenu works along with TEGLUnit to provide the standard drop-down menus and exploding windows. In Chapters 6 through 8, we discuss some of the graphic and mouse primitives that the TEGLUnit uses. You may use some of these routines independently of TEGL. In Chapter 10 we explore the Animation unit along with a sample application that

animates a button icon. Chapter 11 looks at writing text to a window using bit-mapped fonts. Finally, in Chapter 12, we look at the Virtual Memory handler and how to use VM within an application. The Appendices provide greater details on the TEGL Windows Toolkit and the philosophy behind the design.

Frames or Windows?

In this manual the word frame is used often. A frame is our term for the implementation of a window. All the identifiers in the toolkit use frame, not window. You can use these terms interchangeably.

How to Contact TEGL Systems Corporation

If you have any comments or suggestions, you may contact us by writing to

TEGL Systems Corporation
780 - 789 West Pender Street
Vancouver, British Columbia
Canada, V6C 1H2

or phone us at

(604) 669-2577

TEGL Easy

The TEGL Windows Toolkit provides tools to assist you in creating an eye-appealing, functional and intuitive graphical interface to your programs.

There is no fixed format that you must follow when using the TEGL Windows Toolkit. Screen handling, menus, or push button icons are a function of your program design and not a mandatory function of the TEGL Windows Toolkit. However, the tools are provided so you can use emulate the look and feel of most popular windowing packages without locking you into a ridged menu system.

What TEGL Windows Toolkit can do

Overlapping windows are handled without having the application program redraw the window whenever that window is uncovered. This removes the complexity of having to redraw, which is necessary with some windowing systems. The only time a window has to be redrawn is when it is re-sized.

The overhead in maintaining graphic images in memory is offset by the virtual memory manager which automatically swaps the images to EMS and/or disk when more memory is needed. Even with memory swapping, application programs are faster and smaller than those written for other windowing environments.

TEGL handles all mouse and keyboard activities, including all selections of a menu items and clicks on a mouse click area. When the user wants to move a window for instance, the TEGLSupervisor handles all of the user interaction from the clicks of the right mouse button on a window to when the button is released to indicate the new position. When the button is released, and MoveFrameCallProc has been installed for that window, the TEGLSupervisor will call your application procedure with the new location. Your application can either move the frame by calling MoveStackImage or not do so, depending on whatever it determines is appropriate.

Event-Driven Code

While it is possible to write your application in a serial manner using TEGL Windows by polling the keyboard to see if a key has been pressed, or checking the mouse if the mouse has been clicked on an icon or menu, it is much more efficient to write using Event-Driven programming. Event-driven programming is a style of building programs that makes for extremely interactive applications.

An event is simply the automatic calling of one of your application's procedures that is triggered by an action such as the mouse cursor overlapping with an icon on the screen. This type of event handling removes the complex checking of keyboards and mouse devices from the central program and allows for an almost parallel (multitasking) type of

program to be created.

Your choice in programming will determine whether your program responds to the user in a sequential mode where one action must be completed before proceeding to the next, or multiple activities that may be completed at the user's leisure.

A good example of multiple event handling is a program that simulates a calculator. Each key of the calculator pad is tied together with a Mouse Click Area event-handler (ie. a pascal function) that handles that particular key. With the selection of one of the numeric icon keys, the supervisor activates the appropriate event-handler which either adds, multiplies, subtracts, or divides the digits. On completion of the event-handler's task, the control is returned back to the supervisor to await for other events. Other event-handlers, such as notepads, will continue to respond to keyboard or mouse actions along with the activities on the calculator.

An Event is a powerful concept. Hypertext on the MacIntosh is based on a similar structure. By associating an event with a word, image, or icon, you can chain a series of events together. One event may lead to another?

The number of icon/events that can be created is limited only by available memory.

Attaching your Function to an Event

There are six (6) basic types of events that the TEGLSupervisor recognizes. The following will provide a brief discussion on event handling.

{bo Mouse Click Area}

This event occurs whenever the mouse cursor overlaps a defined mouse click area on the screen. Depending on the activation sense, the supervisor may call the event-handler only if the left button is clicked (activation sense set to MsClick), or if the mouse cursor passes over the defined mouse click area (activation sense set to MsSense). The most common use of a mouse click area is the association of an icon with an event-handler.

{bo Click and Drag}

This event is associated with the movement of a window. Control is passed to the Event-handler after a new frame position has been selected. One use of this type of event processing is the dragging of an icon-frame to the trash can (like the MacIntosh).

{bo Expand and Shrink}

This event is associated with the sizing of a window. Control is passed to

the Event-handler after a new frame size has been selected. We use this type of event to re-size a window.

{bo Keyboard Events}

To accommodate systems without a mouse. The Keyboard Event allows you to tie the keyboard to any normal mouse-click-area event handler.

{bo Timer Ticks}

The PC has an internal timer that interrupts the activities of any running program 18 times a second. This interruption is transparent to the operating system and is used mainly to update the system clock.

The TEGL unit uses this timer to provide a flag for the interval of timed events. An event-handler may be defined to occur at resolutions up to 18 times a second or several hours later.

{bo Ctrl-Break}

The Ctrl-Break event is usually tied with the event-handler QUIT, but, like any Event, you may write your own to perform a a different task when a Ctrl-Break event occurs.

Frames

TEGL is a window manager or more correctly a FRAME STACK coordinator. A frame is any defined region of the screen. By stacking two or more frames on the screen, the supervisor monitors the location of the frames and ensures that each frame retains it's own entity.

Once a frame is created, the frame area can be cleared and drawn with any graphic functions provided by the Turbo Pascal language or any other graphical functions provided by other library packages. However, the responsibility of drawing within the window is with the program.

Use the x, y, x1, y1 coordinates provided within the frame record when drawing to the window.

Menus

The TEGL Menus are actually event-handlers that have been written to accommodate drop-down menus, menu selections, lists within a frame, etc.

The menus require a list of items and related events to be created. The list may then be attached to a bar menu using the OutBarOption, which is simply a frame with multiple horizontal mouse click defines.

When TEGLSupervisor senses the mouse overlapping with one of the bar menu selections, an internal BarOptionMenu event is called and a

search is made to find the list that is related to the selection. A menu window is then created and displayed using the list. The menu window is simply another frame with multiple mouse click defines.

A Minimum TEGL Program

The following demo program, prints out the message q Hello World! to a small movable window. Note: this one doesn't require the minimum shell that we described in the Introduction.

```
{ samc0201.pas }

Uses
    dos,
    graph,
    virtmem,
    fastgrph,
    TEGLIntr,
    TEGLWrt,
    TEGLICON,
    TEGLGRPH,
    TEGLUnit,
    TEGLMenu,
    TEGLEasy;


BEGIN
   EasyTEGL;

   PushImage(100,100,200,120);
   Shadowbox(100,100,200,120);
   setcolor(black);
   OutTEGLTextXY(105,105,'Hello World!');

   TEGLSupervisor;
END.
```

Adding Menus (Top Down Design)

A powerful feature in programming with TEGL Windows is the ability to visually see your application develop. Top down design is a methodology

where the layout and menu designs are created first and the functional aspect of the program created later. Program stubs are used as place markers to indicate the required function.

Adding a drop down menu and connecting the event later is a simple task with TEGL Windows.

```
{ samc0202.pas }


VAR
  om1, om2 : OptionMPtr;



FUNCTION GetMsSense(FS:imagestkptr; Ms: msclickptr) : WORD;
  BEGIN
    SetMouseSense(fs^.x,fs^.y);
    GetMsSense := 1;
  END;



BEGIN

   EasyTEGL;


   om1 := CreateOptionMenu(@Font14);
   DefineOptions(om1,' Open ',true,NilUnitProc);
   DefineOptions(om1,'--',false,NilUnitProc);
   DefineOptions(om1,' Quit ',true,Quit);

   om2 := CreateOptionMenu(@Font14);
   DefineOptions(om2,' Memory ',true,ShowCoordinates);
   DefineOptions(om2,' Mouse Sensitivity ',true,GetMsSense);

   CreateBarMenu(0,0,getmaxx);
   OutBarOption(' File ',om1);
   OutBarOption(' Utility ',om2);


   TEGLSupervisor;
END.
```

The events ShowoneFont and ShowFonts are defined in FONTTest, ShowCoordinates and ShowButtonStatus are both defined in

DebugUnt, AskMouseSense is defined in SenseMS, and Quit is defined in TEGLEasy.

ExitOption already exists as a event in the example above.

The rest of the menu selection are all defined to NilUnitProc which is a program event stub that does nothing.

Adding events as you go along is easy, now that the menu is set up.

Adding your First Event

The following is an event that opens a window and writes a message.

```
{ samc0203.pas }
FUNCTION InfoOption(FS:imagestkptr; Ms: MsClickPtr) : WORD;
   VAR
      x,y,x1,y1  : WORD;
      IFS            : ImageStkPtr;
   BEGIN
      Hidemouse;

      x  := 200;
      y  := 120;
      x1 := x+340;
      y1 := y+100;

      PushImage(x,y,x1,y1);
      IFS := StackPtr;

      SetColor(White);
      ShadowBox(x,y,x1,y1);
      SetColor(Black);
      OutTEGLtextxy(x+5,y+5,'TEGL Windows Toolkit II');
      OutTEGLtextxy(x+5,y+5+TEGLCharHeight,
          'Jan 1,1990, Program Written by Richard Tom');

      ShowMouse;

      InfoOption := 1;
   END;
```

Then change the menu declaration line to add InfoOption like so:

```
     DefineOptions(om1,'Info...',TRUE,InfoOption);
```

You may notice that the event returns to the TEGLSupervisor leaving the window on the screen.

We can refined this procedure by adding a while loop to wait for the user to click on a icon. The CheckforMouseSelect(IFS) will return a MouseClickPos once the user has selected the OK icon. While we are changing the event, we might as well add in an expanding and shrinking box effect.

The new event listing.

```
{ samc0204.pas }
FUNCTION InfoOption(FS:imagestkptr; Ms: msclickptr) : WORD;
   VAR
      x,y,x1,y1       : word;
      ifs                 : ImageStkPtr;
      ax,ay,ax1,ay1   : word;
      option          : word;
   BEGIN
      HideMouse;

      x  := 200;
      y  := 120;
      x1 := x+340;
      y1 := y+100;

      ax := Ms^.ms.x+FS^.x;
      ay := Ms^.ms.y+FS^.y;
      ax1 := Ms^.ms.x1+FS^.x;
      ay1 := Ms^.ms.y1+FS^.y;


      PushImage(x,y,x1,y1);
      IFS := stackptr;

      ZipToBox(ax,ay,ax1,ay1,x,y,x1,y1);

      SetColor(White);
      ShadowBox(x,y,x1,y1);

      SetColor(Black);
      OutTEGLtextxy(x+5,y+5,'TEGL Windows Toolkit II');
      OutTEGLtextxy(x+5,y+5+TEGLCharHeight,
        'Jan. 1, 1990, Program Written by Richard Tom');
```

```
    PutPict(x+280,y+75,@ImageOk,Black);
    DefineMouseClickArea(IFS,280,75,280+35,75+12,TRUE,
      NilUnitProc,MSClick);
    SetMousePosition(x+290,y+85);
    ShowMouse;

    WHILE CheckforMouseSelect(IFS)=NIL DO;

    HideMouse;
    DropStackImage(ifs);
    ZipFromBox(ax,ay,ax1,ay1,ifs^.x,ifs^.y,ifs^.x1,ifs^.y1);
    ShowMouse;

    InfoOption := 1;
  END;
```

TEGLEasy

ActiveButton Procedure TEGLEASY

Function

Makes a button/frame.

Declaration

ActiveButton(x,y: Word; s : String; P : CallProc);

Remarks

This is for creating a button which is attacted to a frame that is the same size as the button. P the event can then have as the first statement FrameFromIcon to make a dramatic button to frame transition.

Restrictions

If the ImageStkPtr is required then save the StackPtr immediately after calling ActiveButton.

See also

ExplodeFromIconHide, CollapseToIconShow.

Example

```
 ActiveButton(1,1,'?',HelpEvent);
```

ColToX Function TEGLEASY

Function

Calculates the X coordinate for a text col.

Declaration

ColToX(Col : Integer) : Integer;

Remarks

This is used to treat the graphics display as if it were in text mode to make it easy to place a succession of characters.

Restrictions

The calculation is made using the currently selected font.

See also

RowToY, SetTEGLFont, SetEasyFont.

ErrMess Procedure TEGLEASY

Function

Display an error message.

Declaration

ErrMess(x,y : Word; s : String);

Remarks

The error message s is displayed in a frame at coordinates x,y. The frame is sized to the message and is moved to keep within the confines of the screen.

The frame is displayed until the 'OK' button in the lower right corner is clicked.

See also

GetYesNo.

Example

```
Error(100,100,'You must enter a file name first');
```

FitFrame Procedure TEGLEASY

Function

Creates coordinates that fit on the physical screen.

Declaration

FitFrame(VAR x,y,width,height: Word);

Remarks

x,y are the desired upper left coordinates for a frame. Width and Height are the desired width and height in pixels for the frame. If the starting coordinates would cause the frame to extend beyond the bounds of the screen then they are decremented until the frame will fit. If width or height are greater than their corresponding GetMaxX or GetMaxY then they are set to the maximum screen size.

The lower right coordinates are returned in width=x1, and height=y1.

See also

QuickFrame.

FrameFromIcon Procedure TEGLEASY

Function

Opens a frame in an event that was called from a click on a icon.

Declaration

```
FrameFromIcon(ifs: ImageStkPtr; ms: MsClickPtr;
  x,y,x1,y1 : Word);
```

Remarks

This would be the first statement in an event that is attached to an icon or button created with active button.

This procedure will hide the icon then display an exploding wire box from the icon location to the coordinates x,y,x1,y1 where a frame is opened and cleared. An OK button is placed in the lower right corner of the frame and it is attached to CollapseToIconShow which will close the frame when it is clicked on.

See also

ActiveButton, ExplodeFromIconHide

FrameText Procedure TEGLEASY

Function

Writes text to a frame using row, column coordinates simulating text mode.

Declaration

```
FrameText(ifs : ImageStkPtr; Row,Col : Integer;
  s : String);
```

Remarks

ifs is the frame to write to. Row and Col are the row and column locations relative to the frame. That is, row 1, col 1, is the upper left corner of the frame. Note the coordinates are the reverse of graphics coordinates where column comes first.

Restrictions

The text display is based upon the current font. Swithing fonts will cause uneven text.

Example

```
VAR ifs : ImageStkPtr;
  QuickFrame(ifs,100,100,200,50);
  FrameText(ifs,2,2,'Hello World');
```

GetMousey Function TEGLEASY

Function

Waits for a mouse click and returns the number.

Declaration

```
GetMousey(ifs: ImageStkPtr): Word;
```

Remarks

ifs is the frame where we are waiting for a mouse click to occur. The mouse click number is returned.

GetYesNo Function — TEGLEASY

Function

Gets a yes or no response.

Declaration

GetYesNo(x,y: Word; s : String): Boolean;

Remarks

x,y are the coordinates to display the frame. S is the question to ask, allowing that the only answer can be Yes or No. The frame has a yes and no button displayed in the lower right corner.

This function returns TRUE if Yes is clicked and FALSE if No is clicked.

Example

```
  IF GetYesNo(100,100,'Do you want to erase the file') THEN
    BEGIN
      { -- erase the file }
    END
  ELSE ; { -- cancel the command }
```

EasyTEGL Procedure — TEGLEASY

Function

Does the necessary startup for the toolkit.

Declaration

EasyTEGL;

Remarks

This procedure should be called at the very start of your program. It sets up some default values and clears the screen.

When you have become familiar with the start-up requirements of the TEGL Windows Toolkit then you can write your own initialization procedure.

LastCol Function TEGLEASY

Function

Returns the last column of a frame as if it were in text mode.

Declaration

LastCol(ifs : ImageStkPtr): Integer;

Remarks

The calculation is based upon the currently selected font.

Restrictions

Does not allow for BGI fonts.

See also

LastRow, ColToX, RowToY.

LastRow Function TEGLEASY

Function

Returns the last row of a frame as if it were in text mode.

Declaration

LastRow(ifs : ImageStkPtr): Integer;

Remarks

The calculation is based upon the currently selected font.

Restrictions

Does not allow for BGI fonts.

See also

LastCol, ColToX, RowToY.

OutFrameTextXY Procedure TEGLEASY

Function

Writes text to frame relative coordinates.

Declaration

OutFrameTextXY(ifs : ImageStkPtr; x,y: Word; s : String);

Remarks

Uses the currently selected font.

Restrictions

Does not work with BGI fonts.

See also

FrameText.

Quit Event TEGLEASY

Function

Halts program.

Declaration

Quit(ifs: ImageStkPtr; ms: MsClickPtr): Word;

Remarks

Control break is set to this event by default in EasyTEGL.

```
SetCtrlBreakFS(Quit);
```

QuickFrame Procedure TEGLEASY

Function

Pushes an image and clears the frame.

Declaration

```
QuickFrame(VAR ifs : ImageStkPtr; x,y,width,
  height: Word);
```

Remarks

x,y are the desired upper left coordinates, width and height are the size of the frame. Coordinates are adjusted to fit the physical screen.

After calling QuickFrame the fields x,y,x1,y1 of the ImageStkPtr can be examined to determine the actual frame coordinates.

See also

FitFrame.

Example

```
VAR ifs : ImageStkPtr;

  QuickFrame(ifs,100,100,200,150);
  FrameText(2,2,'This is too TEGL easy!');
```

RestoreFont Procedure TEGLEASY

Function

Restores the current font.

Declaration

RestoreFont;

Remarks

The current font is saved when SelectEasyFont is called.

RowToY Function TEGLEASY

Function

Calculates the Y coordinate for a text row.

Declaration

RowToY(Row : Integer): Integer;

Remarks

This is used to treat the graphics display as if it were in text mode and make it easier to place succeeding rows of text on the screen.

Restrictions

The calculation is based on the current font.

See also

ColToX, LastCol, LastRow, FrameText

SelectEasyFont Procedure TEGLEASY

Function

Changes the font.

Declaration

SelectEasyFont;

Remarks

The font used after this call is selected by previous call to SetEasyFont.

See also

RestoreFont.

SetEasyFont Procedure TEGLEASY

Function

Set the font used by the TEGLEasy Unit.

Declaration

SetEasyFont(p : Pointer);

Remarks

Some of the routines in TEGLEasy write to the screen. This font is used by these routines.

See also

SelectEasyFont, RestoreFont

Example

```
  SetEasyFont(@CountDwn);
```

ICONS

Icons are pictures that represent objects. This Icon image diskdrve represents a diskette.

Icons are the mainstay of GUI's.

The TEGL
Windows Toolkit provides the tools that can create and manipulate icons up to a 100 x 100 pixels in size. By placing an icon within a window frame, they may be attached directly to an TEGL event to provide graphical menu selections, animated to provide visual feedback, displayed as graphic images like the TEGL Deck of Cards, or used to display a company logo.

The ICON Editor

Included in TEGL Windows is a powerful icon editor that utilizes the full power of the tookit to provide you with fast, flexible and easy icon file editing. The source code for the icon editor is also included so you can expand and modify it to suit your needs.

The Main Bar Menu
Open ICONDEF File

Opens an existing ICON.DEF file, or creates a new DEF file. To create a new DEF file, type in the name of the DEF file in the filename box and click on key OK.

Quit
Quits the icon editor. NOTE: The icon editor does not prompt you to save your files.

Editing
The mouse cursor changes to cross-hairs when the cursor enters the icon drawing area. Pressing the mouse left button will place a pixel at the current coordinates. Pressing the mouse right button will erase the pixel. You can hold the mouse left or right button, while moving the mouse to draw or erase a series of pixels.

The drawing bar at the bottom of the drawing area has two functions. Press and hold the right mouse button on the drawing bar to drag the drawing area to a new location. Click with the left mouse button on the drawing bar to select from the drawing menu.

The Drawing Bar Menu

SAVE

Saves the file with the filename displayed on the drawing bar.

SAVE AS

Saves the file with a new filename.

SAVE AND EXIT ICON FILE
Saves the file with the filename displayed on the drawing bar and closes the editing area for the file.

CREATE PASCAL CONSTANTS
Creates a pascal constants file with the extension q .CON for including in a program.

COPY IMAGE AREA

Copies an area into the internal IMAGE AREA. When this option is active a scissors icon appears on the drawing bar. Click once with the left mouse button to mark the upper left corner of the area to copy. Move the mouse cursor to the bottom right corner of the area to copy and click again on the left mouse button. When the scissors disappear, the area has been copied to the internal IMAGE AREA.

CUT IMAGE AREA

Copies an area into an internal IMAGE AREA and clears the Icon area to the background color. When this option is active a scissors icon appears on the drawing bar. Click once with the left mouse button to mark the upper left corner of the area to cut. Move the mouse cursor to the bottom right corner of the area and click again on the left mouse button. When both the scissors disappear and the area is cleared, then the area has been copied to the internal IMAGE AREA.

FILL IMAGE AREA
Fills an area with the current pixel color. Bits that are already set on are not overwritten. When this option is active, a coffee mug icon appears on the drawing bar. Click once with the left mouse button to mark the upper left corner of the area to fill. Move the mouse cursor to the bottom right corner of the area and click again on the left mouse button. The coffee mug disappears when the area is filled with current pixel color.

PASTE IMAGE AREA

Paste the copied/cut area from the internal IMAGE AREA to the icon drawing area. When this option is active, a glue bottle icon appears on the drawing bar. Click once at the position where the image is to be pasted. The pasted image overwrites any pixels on the drawing area.

MERGE IMAGE AREA

Merges the copied/cut area from the internal IMAGE AREA to the icon drawing area. When this option is active, a glue bottle icon appears on

the drawing bar. Click once at the position where the image is to be merged. The merged image only writes to empty pixel areas.

OVERLAY IMAGE AREA

Overlays the copied/cut area from the internal IMAGE AREA to the icon drawing area. When this option is active, a glue bottle icon appears on the drawing bar. Click once at the position where the image is to be overlayed. The overlay image only writes to active pixels.

ROTATE IMAGE AREA 45 DEGREES

Rotates the internal IMAGE AREA by 45 degrees.

ROTATE IMAGE AREA 90 DEGREES

Rotates the internal IMAGE AREA by 90 degrees.

REDUCE IMAGE AREA

Shrinks the image within the internal IMAGE AREA by 50%. The algorithm deletes every second pixel.

REVERSE IMAGE AREA

Reverses the image within the internal IMAGE AREA from left to right.

PIXEL COLOR

Pick the current pixel color from a palette of 16 colors.

BACKGROUND COLOR

Pick the current background color from a palette of 16 colors.

CHANGE PIXELS COLOR

Change all pixels with color m to another color n. Where m and n are selected from a palette of 16 colors. To cancel the command without changing any pixel colors, select the same color for both m and n.

ERASE COLOR PIXELS

Erases all pixels with the selected pixel color. The color is selected from a palette of 16 colors.

EXPLODE ICON IMAGE
Enlarges the drawing area. The largest size is a ratio of 3 to 1 (3 pixels representing 1 pixel).

IMPLODE ICON IMAGE

Shrinks the drawing area.

CLEAR ICON IMAGE
Clears the drawing area.

RELOAD ICON FILE
Reloads the original icon file.

EXIT ICON FILE
Finishes the editing of a icon file.

You can open as many editing windows at once as you like. The internal IMAGE AREA is common to all the edit windows that are open. Consequently, whatever is in the internal IMAGE AREA can be pasted to any edit window. This allows for the building of icons from small pieces, or copying an icon to transform it to something different.

ICON Constants

Select from the Drawing Bar Menu CREATE PASCAL CONSTANTS, to generate constants for including in your program. If you have a large number of icons for generating constants, you can use the program ICONINC to generate all icons in a one pass.

Putpict Procedure FASTGRPH

Function
Puts the defined icon to the specified screen area.
Declaration
Putpict (x,y:word; buf:pointer;n:word)
Remarks
x, y defines the upper left corner of the screen area for placing the icon image.

buf points to the defined icon image.

n defines the color change for any pixel that is black within the icon.

Example

```
const
  ImageMYICON : array [0..1566] of byte =
       ($1D,$06,$83,$01,$5B,  $02,$ ._._._._.
```

```
  PutPict(10,25,@ImageMYICON,black);
```

ICON Assembler Procedures

A drawback of Turbo Pascal is the size of the data area, which limits the number of icons that can be included as constants.

The program ICONASM provides a second method that allows you to add large icon images to your program (eg. the TEGL Deck of Cards).

ICONASM generates a Pascal procedure in assembler. Turbo Assembler is required to assemble the file to object code. You may then create a TPU that will link the icon procedure into your pascal program.

```
procedure ImageMyICON(x,y:word;n:word);
{$L MyIcon.obj}
```

To display the icon, use the icon procedure name (your icon name prefixed with Image).

```
imageMyIcon(10,25,black);
```

Note that these procedures must always be declared as far calls. If you make them part of the interface of a unit then it is done automatically but if you are using any directly in your program or accessing them locally in a unit then be sure and use the {$F+} directive.

ICON Utilities

ICONDEF

ICONDEF is a utility program that allows you to strip the .DEF files from a turbo pascal source file, include file or Assembler file, provided that the commented {.. prefix is still a part of your constants.

Be careful that the Input filename is not the same as one of the definition files. Using a suffix other then .DEF will ensure that the include file is not overwritten while extracting. However, any filenames that do end in .DEF should be copied to a subdirectory if you are not sure about the ICON definition names.

```
  Syntax:  ICONDEF MYFILE.INC

  Where:   MYFILE.INC is the include file generated by ICONINC
           or any file that embeds the include file.
```

ICONLIB

ICONLIB is for assisting the programmer in combining the definition files into a single library file for maintenance. Use ICONDEF to extract.

```
  Syntax:  ICONLIB *[.DEF] MYPROG.DLB

  Where:   *[.DEF] may use any DOS wild-card specifications.
           MYPROG.DLB may be any library filename.
```

ICONINC

ICONINC helps the ICON Editor in generating a large number of Turbo Pascal ICON constants. Multiple icon definitions may be output to a single include file.

```
  Syntax:  ICONDEF *[.DEF] MYFILE.INC

  Where:   *[.DEF] may use any DOS wildcards specifications.
           MYFILE.INC may be any include filename.
```

ICONASM

ICONASM is for assisting the ICON Editor in generating procedures from icon definition files. Multiple procedures may be output to a single asm file.

```
  Syntax:  ICONASM *[.DEF] MYPROG.ASM

  Where:   *[.DEF] may use any DOS wildcards specifications.
           MYPROG.ASM may be any assembler filename.
```

Chapter 3 - Icons

ICONS in TEGLIcon Unit

There are a number of icons that have been created and are available in TEGLIcon unit. You can use these icons by simply including the unit in your USES statement.

ImageCREDITS

TEGL Windows Toolkit II

ImageTRASH
: A trash can

ImageOK
: OK button

ImageCANCEL
: Cancel button

ImageBLANKBUT
: A blank button for creating your own

ImageLBUT ImageMBUT ImageRBUT
: Used by DrawLongButon to create an extra long button icon.

ImageDOWN
: Down arrow.

ImageUP
: Up arrow.

ImageRIGHT
: Right arrow.

ImageLEFT
: Left arrow.

ImageR
: Registered Trademark. reg

ImageC
: Copyright. copyright

ImageTIGER
: A TEGL tiger.

ImageLAST
: Last button.

ImageNEXT
: Next button.

ImageQUESTION
: Question Button.

Chapter 4 - Frames

Frames

The power and speed of TEGL Windows is most apparent when handling frames. By automatically saving and restoring overlapping images, TEGL Windows is a very powerful tool for creating the illusion of separate multiple windows.

Creating, Manipulating, and Dropping Frames

CountFrames Function TEGLUNIT

Function

Returns the number of frames currently on the stack.

Declaration

```
CountFrames: Word;
```

FrameExist Function TEGLUNIT

Function

Determines if a frame is on the frame stack.

Declaration

```
FrameExist(ifs : ImageStkPtr): Boolean
```

Remarks

If ifs exists then it contains the address of one of the frames on the stack.

Example

```
  IF FrameExist(ifs) THEN
    DropStackImage(ifs);
```

PushImage Procedure TEGLUNIT

Function

Used to save the background image before clearing and drawing new images in this area. Equivalent to opening a window area.

Declaration

PushImage(X,Y,X1,Y1 : word)

Remarks

Windows are created by pushing and popping the background image. X, Y, X1, Y1 are absolute coordinates starting with 0,0 at the upper left corner of the screen to GetMaxX, GetMaxY at the lower right corner.

Restrictions

Saving large images can require a lot of memory even with the Virtual Memory Manager. If a program is expected to use most of memory it would be sensible to include specific checks on memory requirements and availability before performing a PushImage.

A full screen in EGA mode (640 x 350) requires about 110K of memory, in VGA mode (640 x 480) the requiment is about 150K.

See also

PopImage, DropStackImage, RotateStackImage, RotateUnderStackImage

Example

The following will create a shadowed box on the upper left screen area. Use the right mouse button to drag the box around.

```
{ samc0401.pas }
PushImage(1,1,100,100);
ShadowBox(1,1,100,100);
```

PopImage Procedure TEGLUNIT

Function

Used to restore the top background image after a PushImage. Equivalent to closing a window area.

Declaration

PopImage

Remarks

Restores the uppermost image area created by PushImage.

See also

PushImage, DropStackImage, RotateStackImage, RotateUnderStackImage

Example

This example waits until a mouse button is pressed then calls PopImage to restore the background image.

```
{ samc0402.pas }
  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);

  WHILE Mouse_Buttons = 0 DO;
  PopImage;
```

RotateStackImage Procedure TEGLUNIT

Function

Rotates a frame forward or backward relative to the frames on the screen.

Declaration

RotateStackImage(var Frame1,Frame2)

Remarks

Frames may be rotated to the foreground to allow user input or updates, etc.

A frame may be rotated as the first frame using RotateUnderStackImage.

In order to access an image that is not the most recent PushImage you must save the Global Variable StackPtr right after the PushImage. The saved

pointer may be used to manipulate the frame.

Restrictions

A frame can only be rotated above a known frame. To rotate a frame below another frame on the stack, use the RotateUnderStackImage routine.

See also

PushImage, PopImage, DropStackImage

Example

The following example creates two overlapping frames and waits for a click of a mouse button before rotating the bottom frame to the top.

```
{ samc0403.pas }
VAR fs : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  FS := stackptr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  RotateStackImage(fs,stackptr);
```

RotateUnderStackImage Procedure TEGLUNIT

Function

Rotates a frame forward or backward relative to the frames on the screen. Rotates a frame below Frame2.

Declaration

RotateUnderStackImage(VAR Frame1,Frame2)

Remarks

In order to access an image that is not the most recent PushImage you must save the Global Variable StackPtr right after the PushImage. The saved pointer may be used to manipulate the frame.

Restrictions

A frame can only be rotated below a known frame. To rotate a frame above another frame on the stack, use the RotateStackImage.

See also

PushImage, PopImage, DropStackImage

Example

The following example creates two overlapping frames and awaits for a click of a mouse button before rotating the Top frame under the second frame.

```
{ samc0404.pas }
VAR FS : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  FS := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  RotateUnderStackImage(StackPtr,fs);
```

DropStackImage Procedure TEGLUNIT

Function

Used to close a frame that is not necessarily the topmost image on the stack. Equivalent to closing a window area.

Declaration

DropStackImage(VAR Frame: ImageStkPtr)

Remarks

Restores an image area created by PushImage.

In order to access an image that is not the most recent PushImage you must save the Global Variable StackPtr right after the PushImage. The saved pointer may be used to manipulate the frame.

See also

PushImage, PopImage, RotateStackImage, RotateUnderStackImage

Example

The following example creates two overlapping frames and awaits for a click of a mouse button before

dropping the bottom frame from the screen.

```
{ samc0405.pas }
VAR fs : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  DropStackImage(fs);
```

HideImage Procedure — TEGLUNIT

Function

Hides an Image Frame from the screen but retains the current stack position and frontal image.

Declaration

HideImage(VAR Frame)

Remarks

This procedure may be used in a variety of ways. Blinking a frame by alternating between HideImage and ShowImage. Moving a frame from one location to another.

See also

ShowImage

Example

The following example blinks a frame continuously until a mouse button is pressed.

```
{ samc0406.pas }
VAR fs : ImageStkPtr;
    i  : word;

  PushImage(1,1,50,50);
  ShadowBox(1,1,50,50);
  fs := StackPtr;
```

```
  i := 20000;
  REPEAT
     dec(i);
     IF i=10000 THEN
        HideImage(fs);
     IF i=0 then
        BEGIN
          ShowImage(fs,fs^.x,fs^.y);
          i := 20000;
        END;
  UNTIL Mouse_Buttons<>0;

  IF i<=10000 THEN
    ShowImage(fs,fs^.x,fs^.y);
```

ShowImage Procedure | TEGLUNIT

Function

Shows a Hidden Image Frame.

Declaration

HideImage(VAR Frame)

See also

HideImage

Example

The following example moves a frame from one location to another when a mouse button is pressed.

```
{ samc0407.pas }
VAR fs : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  HideImage(fs);
```

```
ShowImage(fs,fs^.x+100,fs^.y+100);
```

ShowCoordinates Event DEBUGUNT

Function

A TEGL Event that displays the coordinates of a frame.

Declaration

ShowCoordinates(ifs : ImageStkPtr; Ms : MsClickPtr): Word;

Remarks

This event displays the coordinates of a frame.

Preparing a Frame for Update

PrepareForPartialUpdate Procedure TEGLUNIT

Function

Prepares a portion of a frame for output. Removes all overlapping images above the partial area that is being updated on the screen.

Declaration

PrepareForPartialUpdate(VAR Frame; X,Y,X1,
Y1: word)

Remarks

X,Y,X1,Y1 are absolute coordinates starting with 0,0 at the upper left corner of the screen to GetMaxX, GetMaxY at the lower right corner.

Restrictions

The coordinates must be within the absolute frame coordinates. Use the current Frame coordinates + offsets to obtain the correct absolute coordinates.

PrepareForPartialUpdate and PrepareForUpdate can be used on multiple frames (provided the update areas do not overlap) but must be matched by an equal number of calls to CommitUpdate.

See also

PrepareForUpdate, CommitUpdate

Example

The following example creates two overlapping frames and awaits for a click of a mouse button before drawing a circle on the bottom frame.

```
{ samc0408.pas }
VAR FS,LsPtr : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  PrepareForPartialUpdate(fs,fs^.x,fs^.y,fs^.x1,fs^.y1);
  SetColor(Blue);
  Circle(fs^.x+48,fs^.y+45,40);
  CommitUpdate;
```

PrepareForUpdate Function — TEGLUNIT

Function

Prepares a frame for output. Removes all overlapping images above the frame area that is being updated on the screen.

Declaration

PrepareForUpdate(VAR Frame)

Remarks

Identical to PrepareForPartialUpdate, except the current Frame Coordinates are passed automatically.

Restrictions

PrepareForPartialUpdate and PrepareForUpdate can be used on multiple frames (provided the update areas do not overlap) but must be matched by an equal number of calls to CommitUpdate.

See also

PrepareforPartialUpdate, CommitUpdate

Example

The following example creates two overlapping frames and awaits for a click of a mouse button before drawing a circle on the bottom frame.

```
{ samc0409.pas }
VAR fs : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  PrepareForUpdate(fs);
  SetColor(blue);
  Circle(fs^.x+48,fs^.y+45,40);
  CommitUpdate;
```

CommitUpdate Procedure TEGLUNIT

Function

Commits update. Replaces all overlapping images above the frame area that was being updated on the screen.

Declaration

CommitUpdate;

Remarks

CommitUpdate must be used to close the functions PrepareForPartialUpdate and PrepareForUpdate.

Restrictions

CommitUpdate must be called an equal number of times for each PrepareForPartialUpdate and PrepareForUpdate.

See also

PrepareForPartialUpdate, PrepareForUpdate

Example

The following example creates two overlapping frames and awaits for a click of a mouse button before drawing a circle on the bottom frame.

```
{samc0410.pas }
VAR FS : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  PrepareForUpdate(fs);
  SetColor(blue);
  Circle(fs^.x+48,fs^.y+45,40);
  CommitUpdate;
```

Moving a Frame

FrameSelectAndMove Function TEGLUNIT

Function

Allows a frame to be moved. This routine is normally called by the TEGL supervisor when the right mouse button is held down and the mouse cursor is positioned over a frame.

Declaration

FrameSelectAndMove(mxpos,mypos : word): ImageStkPtr;

Result type

Returns a pointer to the frame that the mouse had selected and moved.

Remarks

The movement of the Frame is under the control of the user until the mouse button is released. To move a frame under program control, use MoveStackImage.

Restrictions

This function returns immediately if neither mouse button is held down on entry.

See also

SetMoveRestrictions, SetFrameMobility,

SetMoveFrameCallProc, MoveStackImage

Example

The following example displays a green mouse cursor and calls FrameSelectAndMove whenever the right mouse button is pressed. The routine exits and changes the mouse cursor back to white when the left mouse button is pressed.

```
VAR FS : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  ShowMouse;
  SetMouseColor(green);
  REPEAT
    IF Mouse_Buttons=2 THEN
      fs := FrameSelectAndMove(Mouse_Xcoord,Mouse_Ycoord);
  UNTIL Mouse_Buttons = 1;
  SetMouseColor(white);
```

SetAutoRotate Procedure — TEGLUNIT

Function

Sets the frame stack auto rotate function.

Declaration

SetAutoRotate(OnOff: Boolean);

Remarks

Auto rotate is normally set to FALSE. That is, a frame will not automatically rotate to the top of the stack. When set to TRUE any frame that is partially covered will be moved to the top of the stack when TEGLSupervisor detects a left mouse button click anywhere on the frame.

Example

```
  { -- after this frames jump to the top with a click of the mouse }
  SetAutoRotate(TRUE);
```

SetMoveRestrictions Procedure TEGLUNIT

Function

Sets the minimum and maximum coordinates that a frame may be moved.

Declaration

SetMoveRestrictions(sh VAR frame; x,y,x1,y1: word)

Remarks

Sets the area that a frame is restricted to when FrameSelectAndMove is called.

Restrictions

The restriction does not apply when a frame is moved using MoveStackImage.

See also

FrameSelectAndMove, SetFrameMobility, SetMoveFrameCallProc, MoveStackImage

Example

The following sets the frame mobility to the upper half of the screen. Use the right mouse button to move the frame around.

```
PushImage(1,1,100,100);
ShadowBox(1,1,100,100);
SetMoveRestrictions(StackPtr,0,0,GetmaxX,GetmaxY div 2);
```

SetFrameMobility Procedure TEGLUNIT

Function

Toggles the ability for a frame to move.

Declaration

SetFrameMobility(sh VAR frame; movable: boolean)

Remarks

When the mobility of a frame is set to off (false), the frame outline will move when FrameSelectAndMove is called, however, the frame is not moved to the new location when the mouse button is released.

The default frame mobility is ON (true).

Restrictions

The mobility toggle has no effect when a frame is moved using MoveStackImage.

See also

FrameSelectAndMove, SetMoveRestrictions, SetMoveFrameCallProc, MoveStackImage

Example

The following example toggles a frames mobility to off.

```
  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  SetFrameMobility(StackPtr,false);
```

SetMoveFrameCallProc Procedure — TEGLUNIT

Function

An event process that is called after an frame has been dragged to a new screen position.

Declaration

```
SetMoveFrameCallProc(sh VAR frame : ImageStkPtr;
  P : CallProc);
```

Remarks

Can be used for the trash can effect, originating with the MacIntosh, by which file icons are dragged to the trash can to be deleted from the system.

The event may check the MouseClickPos Record (fields MS.X, MS.Y, MS.X1, and MS.Y1) for the new frame location and whether they overlap the desired frame.

Restrictions

If you wish for the frame to move to the new location, the event must call MoveStackImage before returning.

See also

FrameSelectAndMove, SetMoveRestrictions, SetFrameMobility, MoveStackImage

Example

The following is a very simple Event Handler that simply closes the frame if the frame is moved.

```
Function Poof(Frame:ImageStkPtr; MouseClickPos : MsClickPtr) : Word;
   BEGIN
      HideMouse;
      DropStackImage(frame);
      ShowMouse;
      Poof := 0;
   END;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  SetMoveFrameCallProc(StackPtr,Poof);
```

MoveStackImage Procedure — TEGLUNIT

Function

Move a frame to a new screen location.

Declaration

MoveStackImage(sh VAR Frame; x,y : word)

Remarks

Used to move a frame under Program control to a new screen location. X and Y are absolute coordinates that specify the upper left corner of the frame at the new location.

Restrictions

The coordinates are not validated, so care must be taken to ensure that the resulting coordinates of the lower right corner falls within the screen area.

See also

FrameSelectAndMove, SetMoveRestrictions, SetFrameMobility, SetFrameCallProc

Example

The following example moves a smaller frame under another larger frame to demonstrate the integrity of stacked images.

```
VAR fs : ImageStkPtr;
    i  : word;
```

```
PushImage(1,1,20,20);
ShadowBox(1,1,20,20);
fs := StackPtr;

PushImage(50,50,150,150);
ShadowBox(50,50,150,150);

FOR i:=1 to 100 DO
  MoveStackImage(fs,fs^.x+2,fs^.y+2);
```

MoveFrame Procedure TEGLUNIT

Function

Moves an Xor wire frame from one location to another.

Declaration

```
MoveFrame(VAR fx,fy,fx1,fy1 : Integer;
  rx,ry,rx1,ry1: Integer; Color: Integer);
```

Remarks

This only moves a wire frame not the actual frame. The mouse button must be held down on entry or this function returns immediately. rx,ry,rx1,ry1 are the starting coordinates. fx,fy,fx1,fy1 are the coordinates when the mouse button is released. Color is the wireframe color.

Low Level Frame Functions

UnLinkFS Procedure TEGLUNIT

Function

Disconnects a frame from the stack.

Declaration

```
UnLinkFS(sh VAR Frame)
```

Remarks

UnLinkFS allows you to disconnect a frame from the Image stack to stop any further actions by the frame manager.

This procedure is used throughout the window management routines. It is provided as an external routine only for specialized needs.

Restrictions

This procedure should be used in conjunction with HideImage, ShowImage, CreateImageBuffer, DropImageBuffer, and LinkFS.

If you unlink a frame from the stack without first hiding the frame, the stack manager will not acknowledge the existence of the frame and will overwrite the unlinked frame area.

See also

LinkFS, LinkUnderFS

Example

The following example hides the frame before unlinking and dropping the image.

```
VAR FS : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  HideImage(fs);
  UnLinkFS(fs);
  DropImageBuffer(fs);
```

LinkFS Procedure — TEGLUNIT

Function

Reconnects a frame to the stack.

Declaration

LinkFS(sh VAR Frame1,Frame2)

Remarks

LinkFS reconnects Frame1 with the Frame stack, above Frame2.

This procedure is used throughout the window management routines. It is provided as an external routine only for specialized needs.

Restrictions

This procedure should be used in conjunction with HideImage, ShowImage, CreateImageBuffer, DropImageBuffer, and UnLinkFS.

See also

UnLinkFS, LinkUnderFS

Example

The following example performs the same function as RotateStackImage.

```
VAR FS : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);

  WHILE Mouse_Buttons = 0 DO;

  HideImage(fs);
  UnLinkFS(fs);
  LinkFS(fs,StackPtr);
  ShowImage(fs,fs^.x,fs^.y);
```

LinkUnderFS Procedure TEGLUNIT

Function

Reconnects a frame with the frame stack, below the specified frame.

Declaration

LinkUnderFS(sh VAR Frame1,Frame2)

Remarks

LinkUnderFS reconnects Frame1 below Frame2.

This procedure is used throughout the window management routines. It is provided as an external routine only for specialized needs.

Restrictions

This procedure should be used in conjunction with HideImage, ShowImage, CreateImageBuffer, DropImageBuffer, and UnLinkFS.

See also

UnLinkFS, LinkFS

Example

The following example performs the same function as RotateUnderStackImage.

```
VAR fs1,fs2 : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs1 := StackPtr;

  PushImage(50,50,150,150);
  ShadowBox(50,50,150,150);
  fs2 := StackPtr;

  WHILE Mouse_Buttons = 0 DO;

  HideImage(fs2);
  UnLinkFS(fs2);
  LinkUnderFS(fs2,fs1);
  ShowImage(fs2,fs2^.x,fs2^.y);
```

CreateImageBuffer Procedure TEGLUNIT

Function

Allocates an Image buffer (frame) on the Heap.

Declaration

CreateImageBuffer(VAR Frame; x,y,x1,y1:word)

Remarks

This procedure is used throughout the window management routines. It is provided as an external routine only for specialized needs.

Restrictions

This procedure should be used in conjunction with HideImage, ShowImage, CreateImageBuffer, DropImageBuffer, and UnLinkFS.

See also

DropImageBuffer

Example

The following example performs the same function as PushImage.

```
VAR FS : ImageStkPtr;

  CreateImageBuffer(fs,1,1,100,100);
  LinkFs(fs,StackPtr);
  GetBiti(1,1,100,100,fs^.imagesave);

  ShadowBox(1,1,100,100);
```

DropImageBuffer Procedure TEGLUNIT

Function

Frees the memory used by the frame on the heap.

Declaration

DropImageBuffer(VAR Frame)

Remarks

This procedure is used throughout the window management routines. It is provided as an external routine only for specialized needs.

Restrictions

This procedure should be used in conjunction with HideImage, ShowImage, CreateImageBuffer, DropImageBuffer, and UnLinkFS.

See also

CreateImageBuffer

Example

The following example performs the same function as PopImage.

```
VAR FS : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  WHILE Mouse_Buttons = 0 DO;

  UnlinkFS(fs);
  DropImageBuffer(fs);
```

GetFSImage Function — TEGLUNIT

Function

Retrieves the screen image within a stacked frame.

Declaration

GetFSImage(Frame)

Result type

Returns a (non-stacked) frame containing the screen image and other related frame information.

Remarks

The (non-stacked) frame may be used for replication or it can be merged with other frames.

See also

PutFSImage

Example

The following example creates a single frame and replicates the frame.

```
VAR FS,TS : ImageStkPtr;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  ts := GetFSImage(fs);
  PushImage(51,51,150,150);
  PutFSImage(51,51,ts,FGNORM);
  DropImageBuffer(ts);
```

PutFSImage Procedure TEGLUNIT

Function

Places the frame saved image anywhere on the screen.

Declaration

PutFSImage(x,y,Frame,RWBITS)

Remarks

RWBITS are constants defined in EGAGRAPH which defines how the images are placed on the screen.

FGNorn

replaces screen area with frame image

FGAnd

AND's screen area with frame image. Toggles off screen areas that do no have a frame image. Creates an outline of the frame image.

FGOr

OR's screen area with frame image. Toggles on empty screen areas that have a frame image. Creates a solid frame image.

FGXor

XOR's screen area with frame image.

FGNot

Inverts frame image and replaces screen area with image.

See also
GetFSImage
Example

The following example creates a single frame and replicates the frame.

```
VAR fs,ts : ImageStkPtr;
    i    : word;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  fs := StackPtr;

  ts := GetFSImage(fs);

  FOR i:=1 to 20 DO
```

```
    BEGIN
      PushImage(1+i*10,1+i*10,100+i*10,100+i*10);
      PutFSImage(1+i*10,1+i*10,ts,FGNOT);
    END;

  DropImageBuffer(ts);
```

FreeImageBuffer Procedure — TEGLUNIT

Function

Frees up the memory allocated for a frame buffer.

Declaration

```
FreeImageBuffer(VAR ifs : ImageStkPtr);
```

Remarks

This is generally an internal function. Do not use it unless you have a clear understanding of inner workings of the frame stack.

GetPartialFrontImage Function — TEGLUNIT

Function

Gets the partial image of a frame and returns a pointer to a temporary buffer.

Declaration

```
GetPartialFrontImage(Frame: ImageStkPtr;
  x,y,x1,y1 : Word) : ImageStkPtr;
```

Remarks

This is a safer way to get the partial image of a frame than using GetBiti. Overlapping frames are partially removed and then restored before returning.

GetFrontImage Function TEGLUNIT

Function

Get the image of a frame and returns a pointer to a temporary buffer.

Declaration

GetFrontImage(ifs : ImageStkPtr): ImageStkPtr;

Remarks

This is a safer way to get the image of a frame than using GetBiti. Overlapping frames are partially removed and then restored before returning.

PageInFS Procedure TEGLUNIT

Function

Read an image into memory.

Declaration

PageInFS(VAR ifs : ImageStkPtr);

Remarks

If the image is already in memory then no action is taken.

See also

PageOutFS.

Example

This example checks to see if the image is in memory first before attempting to read it in. Note that PageInFS check this automatically before reading in an image.

```
  IF ifs^.ImagePageOut THEN    { -- the image is not in memory }
    PageInFS(ifs);
```

LockImage Procedure TEGLUNIT

Function

Locks an frame image into memory.

Declaration

LockImage(VAR ifs : ImageStkPtr);

Remarks

The image is read into memory if required. The lock is maintained until a specific call is made to UnLockImage.

Lock image can be used where it is desirable to replicate an image on the screen repeatedly. After it is locked then it can be placed on the screen with a call to PutBiti.

Restrictions

This should be used with caution especially if you are locking in a large image. You can fragment the heap and the Virtual Memory Manager may not be able to allocate a large enough memory block for subsequent image swaps.

See also

UnLockImage, UseImage, UnUseImage

Example

If the image is less than 64k then it can be copied to Turbo's heap and then the image can be unlocked reducing the chance of a heap error.

```
VAR ifs : ImageStackPtr;
    buf : Pointer;

  PushImage(100,100,300,150);
  ifs := StackPtr;
  ShadowBox(100,100,300,150);
  { -- do something with the frame }
  { -- then lock it so its not swapped out }
  LockImage(ifs);
  { -- allocate memory on Turbo's Heap }
  GetMem(buf,ifs^.ImageSize);
  { -- move it there }
  buf^ := ifs^.imagesave;
  { -- unlock the image }
  UnLockImage(ifs):
```

PageOutFS Procedure — TEGLUNIT

Function

Page out a frame image.

Declaration

PageOutFS(VAR ifs : ImageStkPtr);

Remarks

If the image is successfully paged out to EMS or disk then TEGLFreeMem is called to free up the memory used.

Restrictions

If ifs is in use, or locked or already paged out then no action is taken.

See also

PageInFS.

Example

```
  PageOutFS(ifs);
  IF ifs^.ImagePageOut THEN { -- success }
    ELSE ; { -- failure }
```

SetImageCoordinates Procedure — TEGLUNIT

Function

Sets the frame pointer to a new set of coordinates.

Declaration

```
SetImageCoordinates(VAR ifs : ImageStkPtr;
  x,y,x1,y1 : Word);
```

Remarks

A frame's coordinates should not be changed if it is visible.

PageOutImageStack Function — TEGLUNIT

Function

Requests the virtual memory manager to page out images to make a chunk of memory available.

Declaration

PageOutImageStack(Mem : LongInt) : Boolean

Remarks

Mem is the amount of memory required. A large value for Mem will result in all image buffers being paged out. This function returns true if the amount of memory requested has been freed.

Restrictions

Large amounts of memory are required to process image swapping. If you allocate too much and don't free it up as quickly as possible you may get a heap error.

Example

```
{ -- force all imagebuffers to disk }
IF PageOutImageStack(512000) THEN;   { -- ignore result }
{ -- do whatever needs that much memory }
SuperSortMemUse(MaxAvail);
SuperSort;
{ -- release it before working with windows again }
SuperSortFreeMem;
```

UnLockImage Procedure — TEGLUNIT

Function

Unlocks a frame image.

Declaration

UnLockImage(VAR ifs : ImageStkPtr);

Remarks

Unlock simply sets a flag in the ImageStkPtr. After unlocking, the Virtual Memory Manager can swap the image to EMS or Disk as required. If the

image wasn't locked then no action is taken.

Restrictions

See restrictions for LockImage.

See also

LockImage, UseImage, UnUseImage.

Example

See example for LockImage.

UnUseImage Procedure TEGLUNIT

Function

Flags a frame image as no longer in use.

Declaration

UnUseImage(VAR ifs : ImageStkPtr);

Remarks

This should be called as soon as possible after a UseImage to keep as much memory free for the Virtual Memory Manager.

See also

UseImage, LockImage, UnLockImage.

Example

```
  UseImage(ifs);
  { -- do something with it }

  { -- then let the memory manager swap it out if required }
  UnUseImage(ifs);
```

UseImage Procedure TEGLUNIT

Function

Makes an image available for use.

Declaration

UseImage(VAR ifs : ImageStkPtr);

Remarks

The frame image is read into memory if not already then and then flagged as being in use.

Restrictions

If you do PrepareForUpdate then the in use flag is set to false.

See also

UnUseImage, LockImage, UnLockImage.

Example

```
UseImage(ifs);
{ -- do something with it }

{ -- then let the memory manager swap it out if required }
UnUseImage(ifs);
```

Mouse Click Areas

Mouse click areas are those places on the screen where we sense if the mouse pointer has passed over or has been clicked on. Frames can have mouse click areas on them that are, of course, only available if the frame is visible and the mouse click area is uncovered.

The sensitivity of the mouse click area has two levels. The most sensitive is MsSense where just having the mouse pointer pass over the area causes an action. The other level is MsClick where the mouse pointer must be over the mouse click area and the left mouse button has been pressed.

DefineMouseClickArea Procedure   TEGLUNIT

Function

Attaches an sensitive area of a frame to an event function.

Declaration

DefineMouseClickArea(VAR ifs : ImageStkPtr; x,y,x1,y1: Integer; Active : Boolean; P : CallProc, Sense: Boolean);

Remarks

ifs is any ImageStkPtr. The x, y, x1, y1 are coordinates relative to a frame. This means that the upper left corner of a frame is considered 0,0.

Active is a boolean flag to indicate whether the Mouse Click Area is an active entry True or a place holder False in a list of mouse clicks. A place holder is simply a defined entry with no action recognized.

p is the event to call when the Mouse Click Area is activated, either by the mouse pointer passing by the click areas or a mouse click occurring on an click area.

NilUnitProc may be used to define a no-event handler. This may be used in conjunction with the functions FindFrame and CheckMouseClickPos to check for the respective mouse click activation.

NilUnitProc may also be used as a temporary parameter. Use ResetMSClickCallProc to add the proper event handler later.

Sense is either MSSense or MSClick. MSSense activates the event handler whenever the mouse cursor passes over the defined mouse click areas. MSClick requires the right mouse button to be pressed while the mouse cursor is on the mouse click area.

Restrictions

The number of mouse click areas is limited only by memory. Overlapping click area take priority over underlying click areas.

The coordinates of a Mouse click area must reside within the Frame, otherwise the click areas are not recognized.

See also

FindMouseClickPtr, ResetMouseClicks, ResetMSClickSense, ResetMSClickCallProc, ResetMSClickActive, CheckMouseClickPos

Example

The following example creates a frame that attaches an 'OK' icon with an Event Handler called DropBoxOption which simply closes the frame and exits.

The function CheckforMouseSelect is used to create the illusion of a button being pressed when clicked on.

```
Function DropBoxOption(Frame:ImageStkPtr; MouseClickPos: MSClickPtr):WORD;
```

```
   BEGIN
      IF CheckforMouseSelect(Frame)<>nil then
        BEGIN
           Hidemouse;
           DropStackImage(Frame);
           ShowMouse;
        END;

      DropBoxOption := 0;
   END;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  PutPict(50,80,@ImageOk,black);
  DefineMouseClickArea(StackPtr,50,80,50+35,80+12,true,
    DropBoxOption,MSClick);
```

FindMouseClickPtr Function — TEGLUNIT

Function

Searches for a Mouse Click Pointer associated with a Mouse Click Number.

Declaration

FindMouseClickPtr(VAR ifs : ImageStkPtr; Clicknumber: Word);

Result type

Returns a mouse click pointer (MSClickPtr), pointing to a Mouse Click Record.

Remarks

Click Numbers are in the order that you define the Mouse Click areas. The first DefineMouseClickArea is known as Click Number 1, the second is Click Number 2, etc..

In certain instances it is easier to advance through the mouse click areas by Click Numbers. However, most functions, including the calling of Events, pass the Mouse Click Pointer.

To translate a Mouse Click Pointer back to a Click Number, use the Mouse Click Pointer fields ie. ClickNumber := MouseClickPos^.ClickNumber where MouseClickPos is of type MSClickPtr.

Restrictions

FindMouseClickPtr returns a Nil if the clicknumber is not found. Compare the resulting MSClickPtr with Nil before referencing the record.

See also

DefineMouseClickPtr, ResetMouseClicks, ResetMSClickSense, ResetMSClickCallProc, ResetMSClickActive, CheckMouseClickPos

Example

The following example defines an array of 100 Mouse Click Areas. You may click with the left mouse button on the individual tiles to produce a sound, or on the 'OK' to produce a series of sounds.

The function FindMouseClickPtr is used within the event handler PlayAllNotes to translate a random click number into a note.

The function CheckforMouseSelect is used to create the illusion of a button being pressed when clicked on.

```
VAR x,y : word;

Function PlayOneNote(Frame:ImageStkPtr;
           MouseClickPos: MSClickPtr):WORD;
   BEGIN
      ToggleOptionBar(Frame,MouseClickPos,nil);
      Beep(MouseClickPos^.clicknumber*10,1,100);
      ToggleOptionBar(Frame,nil,MouseClickPos);
      PlayOneNote := 0;
   END;

Function PlayAllNotes(Frame:ImageStkPtr;
           MouseClickPos: MSClickPtr):WORD;
   VAR i,rs : word;
   BEGIN
      IF CheckforMouseSelect(Frame)<>nil THEN
        BEGIN
           FOR i:=1 to 30 DO
              rs := PlayOneNote(Frame,FindMouseClickPtr(Frame,
                random(100)+1));
        END;
      PlayAllNotes := 0;
   END;

  PushImage(1,1,107,124);
  ShadowBox(1,1,107,124);

  FOR x:=0 to 9 DO
    FOR y:=0 to 9 DO
```

```
      BEGIN
        ShadowBox(StackPtr^.x+3+x*10,StackPtr^.y+3+y*10,
          StackPtr^.x+3+10+x*10,StackPtr^.y+3+10+y*10);
        DefineMouseClickArea(StackPtr,3+x*10,3+y*10,
          3+x*10+6,3+y*10+6,true,PlayOneNote,MSClick);
      END;

  Putpict(StackPtr^.x+50,StackPtr^.y+105,@imageok,black);
  DefineMouseClickArea(StackPtr,50,105,50+35,105+12,true,
    PlayAllNotes,MSClick);
```

ResetMSClickActive Procedure TEGLUNIT

Function

Resets the active flag to indicate whether a Mouse Click Area Entry is active or inactive.

Declaration

```
ResetMSClickActive(VAR ifs: ImageStkPtr;
  MouseClickNumber : Word; Active : Boolean);
```

Remarks

The MouseClickNumber is in the order that you defined the Mouse Click areas. The first DefineMouseClickArea is known as MouseClickNumber 1, the second is MouseClickNumber 2, etc..

Active is a boolean flag to indicate whether the Mouse Click Area is an active entry (True) or a place holder (False) in a list of mouse clicks. A place holder is simple a defined entry with no action recognized.

Restrictions

If the MouseClickNumber is invalid, the flag is not updated.

See also

DefineMouseClickPtr, ResetMouseClicks, FindMouseClickPtr, ResetMSClickSense, ResetMSClickCallProc, CheckMouseClickPos

Example

This example creates an array of 10 buttons which all point to the same Event Handler SwitchOn. The active flag for a pressed button is turned off to prevent multiple calls to SwitchOn, until another button is pressed. ResetMSClickActive is used within SwitchOn to toggle the button Active state.

```
VAR x,y : word;

function SwitchOn(Frame:ImageStkPtr;
           MouseClickPos: MsClickPtr) : word;
   VAR i  : word;
       ms : msclickptr;
   BEGIN
      Beep(1500,1,10);

      FOR i:=1 to Frame^.MSClickCount DO
        BEGIN
          MS := FindMouseClickPtr(Frame,i);
          IF NOT MS^.MSActive THEN
            BEGIN
              HideMouse;
              PutPict(Frame^.x+MS^.ms.x,Frame^.y+MS^.ms.y,
                @imageBlankBut,black);
              ResetMSClickActive(Frame,MS^.ClickNumber,true);
              ShowMouse;
            END;
        END;

      HideMouse;
      PutPict(Frame^.x+MouseClickPos^.ms.x,
        Frame^.y+MouseClickPos^.ms.y,@ImageOk,black);
      ShowMouse;
      PressButton(Frame,MouseClickPos);
      ResetMSClickActive(Frame,MouseClickPos^.ClickNumber,false);

      SwitchOn := 1;
   END;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);

  FOR x:=0 to 1 DO
     FOR y:=0 to 4 DO
        BEGIN
          Putpict(StackPtr^.x+6+x*42,StackPtr^.y+6+y*18,
            @imageBlankBut,black);
          DefineMouseClickArea(StackPtr,5+x*42,5+y*18,
            5+x*42+35,5+y*18+12,true,SwitchOn,MSClick);
        END;
```

ResetMSClickCallProc Procedure TEGLUNIT

Function

Changes the Event Handler for a Mouse click to another Event Handler.

Declaration

ResetMSClickCallProc(ifs : ImageStkPtr; MouseClickNumber: Word; P : CallProc);

Remarks

MouseClickNumbers are in the order that you define the Mouse Click areas. The first DefineMouseClickArea is known as MouseClickNumber 1, the second is MouseClickNumber 2, etc..

p is the event to pass control to when the mouse click area is actived.

NilUnitProc may be used to define a no-event handler. This may be used in conjunction with the functions FindFrame and CheckMouseClickPos to check for the respective mouse click activation.

NilUnitProc may also be used to deactivate an event handler.

See also

DefineMouseClickPtr, ResetMouseClicks, FindMouseClickPtr, ResetMSClickSense, ResetMSClickActive, CheckMouseClickPos

Example

This example switches between two events that play a different series of sounds. The function CheckforMouseSelect is used to create the illusion of a button being pressed when clicked on.

```
function FirstSong(Frame:ImageStkPtr;
           MouseClickPos: MsClickPtr): word; FORWARD;

function SecondSong(Frame:ImageStkPtr;
           MouseClickPos: MsClickPtr): word;
   BEGIN
      IF CheckforMouseSelect(Frame)<>nil THEN
        BEGIN
           Beep(1500,5,100);
           ResetMSClickCallProc(frame,
             MouseClickPos^.ClickNumber,FirstSong);
        END;
      SecondSong := 1;
```

```
   END;

function FirstSong(Frame:ImageStkPtr;
           MouseClickPos: msclickptr) : word;
   BEGIN
      IF CheckforMouseSelect(Frame)<>nil THEN
        BEGIN
           SlideBeep(500,1500,2);
           ResetMSClickCallProc(frame,
             MouseClickPos^.ClickNumber,SecondSong);
        END;
      FirstSong := 1;
   END;

  PushImage(1,1,100,100);
  ShadowBox(1,1,100,100);
  Putpict(StackPtr^.x+51,StackPtr^.y+81,@ImageOk,black);
  DefineMouseClickArea(StackPtr,50,80,50+35,80+12,
    true,FirstSong,MSClick);
```

ResetMouseClicks Procedure — TEGLUNIT

Function

Removes a chain of mouse click areas from a frame.

Declaration

ResetMouseClicks(Frame,ClickPtr:MSClickPtr)

Remarks

The ClickPtr parameter is the last click pointer from where the remainder of the chain of click areas will be removed.

A parameter of Nil removes the Mouse Click Area chain completely.

Restrictions

The ClickPtr should be a valid Mouse Click Ptr. Use FindMouseClickPtr to locate a valid pointer.

If ClickPtr is invalid, the parameter will be treated as Nil.

See also

DefineMouseClickPtr, FindMouseClickPtr, ResetMSClickSense, ResetMSClickCallProc, ResetMSClickActive, CheckMouseClickPos

Example

The following example displays a varying number of bars that can be selected. The Event Handler ShowBarList plays a sound corresponding to the bar selected and clears the frame and re-displays a new series of bars.

```
Function ShowBarList(Frame:ImageStkPtr;
           MouseClickPos: MSClickPtr):WORD; FORWARD;

Procedure ShowVaryList(fs:ImageStkPtr; N:word);
   VAR y : word;
   BEGIN
      ResetMouseClicks(fs,nil);
      FOR y:=0 to n DO
        BEGIN
           ShadowBox(fs^.x+5,fs^.y+3+y*10,fs^.x1-8,fs^.y+3+10+y*10);
           DefineMouseClickArea(StackPtr,5,3+y*10,
             fs^.x1-fs^.x-10,3+y*10+6,true,ShowBarList,MSClick);
        END;
   END;

Function ShowBarList(Frame:ImageStkPtr;
           MouseClickPos: MSClickPtr):WORD;
   BEGIN
      ToggleOptionBar(Frame,MouseClickPos,nil);
      Beep(MouseClickPos^.clicknumber*30,10,100);
      HideMouse;
      ShadowBox(frame^.x,frame^.y,frame^.x1,frame^.y1);
      ShowVaryList(frame,random(10)+1);
      ShowMouse;
   END;

  PushImage(1,1,107,124);
  ShadowBox(1,1,107,124);
  ShowVaryList(StackPtr,random(10)+1);
```

ResetMSClickSense Procedure TEGLUNIT

Function

Resets the Sense parameter associated with a Mouse

Click Area.

Declaration

ResetMSClickSense(VAR ifs : ImageStkPtr; NewSense : Boolean;)

Remarks

NewSense is either MSSense or MSClick. MSSense activates the event handler whenever the mouse cursor passes over the defined mouse click areas. MSClick requires the right mouse button to be pressed while the mouse cursor is on the mouse click area.

Restrictions

ResetMSClickSense resets the Sense type for the chain of all Mouse Clicks. If you have a mixture of different senses, use a combination of FindMouseClickPtr and field settings to reset the sense.

See also

DefineMouseClickPtr, ResetMouseClicks, FindMouseClickPtr, ResetMSClickCallProc, ResetMSClickActive, CheckMouseClickPos

Example

The following example requires a menu selection to toggle between the menu dropping down automatically or requiring a mouse clickon the menu bar.

```
VAR  OM1          : OptionMptr;
     ToggleSense : boolean;

Function ToggleClickSense(Frame:ImageStkPtr;
           MouseClickPos: MsClickPtr) : word;
   VAR MenuBarFS   : ImageStkPtr;
   BEGIN
      MenubarFS := Frame^.RelatedStack;

      ToggleSense := NOT ToggleSense;
      ResetMSClickSense(MenubarFS,ToggleSense);

      ToggleClickSense := 1;
   END;

  OM1 := CreateOptionMenu(@FONT14);
  DefineOptions(OM1,'Toggle Click Sense',true,ToggleClickSense);

  FONTTABLE := @FONT14;
  CreateBarMenu(0,0,639);
    OutBarOption(' ToggleBar  ',OM1);
    OutBarOption(' ToggleTwo  ',OM1);
  ToggleSense := MSSense;
```

Keyboard

ClearKeyBoardBuf Procedure TEGLUNIT

Function
: Clears the hardware keyboard buffer.

Declaration
: ClearKeyBoardBuf;

See also
: ClearTEGLKeyBoardBuf.

ClearTEGLKeyBoardBuf Procedure TEGLUNIT

Function
: Clears the software buffer maintained by the Toolkit.

Declaration
: ClearTEGLKeyBoardBuf;

Remarks
: This will discard all pending keystrokes.

DefineGlobalKeyClickArea Procedure TEGLUNIT

Function
: Flexible keycode assignment.

Declaration
: DefineGlobalKeyClickArea(ifs : ImageStkPtr;

```
  ms : MsClickPtr; KeyCode : Word; RepeatKey: Boolean;
    p : CallProc);
```

Remarks

ifs is the frame and ms is the mouse click area the key is assigned to, these are passed to p.

If ifs and ms are set to nil then the frame and mouse click area that the mouse pointer is over are passed to p. If the mouse pointer is not over a frame then Nil is passed to p.

If RepeatKey is set True then addition key presses are buffered, otherwise, they are discarded.

A special case for this routine is passing 0 as the keycode parameter. In this case any key that is not being trapped for will activate p. The key pressed can be determined by using ReadKey.

Restrictions

Only the most recently declared key is trapped if a key is trapped more than once.

See also

DefineLocalKeyClickArea.

DefineLocalKeyClickArea Procedure — TEGLUNIT

Function

Assign a keycode to a frame and mouse click area.

Declaration

```
DefineLocalKeyClickArea(fs : ImageStkPtr;
  ms : MsClickPtr; KeyCode : Word; RepeatKey: Boolean:
    p : CallProc);
```

Remarks

ifs is the frame and ms is the mouse click area the key is assigned to, these are passed to p.

If RepeatKey is set TRUE then addition key presses are buffered otherwise they are discarded.

Within a frame DefineLocalKeyClickArea has prioritry over DefineGlobalKeyClickArea.

See also

DefineGlobalKeyClickArea.

DropKeyClick Procedure TEGLUNIT

Function

Removes a key trap.

Declaration

```
DropKeyClick(ifs : ImageStkPtr; KeyCode: Word;
  p : CallProc):
```

Remarks

If ifs is not Nil then the frame's local key stack is searched first. If the key is not found then the search proceeds to the global key stack.

p must match the CallProc that the key was originally assigned to.

FindKeyClickPtr Function TEGLUNIT

Function

Locates a key assignment.

Declaration

```
FindKeyClickPtr(ifs : ImageStkPtr; Keycode: Word) :
  KeyClickPtr;
```

Remarks

If ifs is not Nil then the frame's local key stack is searched first. If the key is not found then the search proceeds to the global key stack KeyStackPtr.

If the KeyCode is not found then NIL is returned.

ResetKeyClickCallProc Procedure TEGLUNIT

Function

Changes the CallProc a key is assigned to.

Declaration

```
ResetKeyClickCallProc(ifs : ImageStkPtr; Keycode: Word;
  p : CallProc);
```

Remarks

If ifs is not NIL then the frame's local key stack is searched first. If the key is not found then the search proceeds to the global key stack KeyStackPtr.

If KeyCode is not found then no action is taken.

Chapter 5 - Menus

Drop Down, Pop Up Menus

The Menu unit is good example of an event library that you can add to the power of TEGL Windows. The generic pull-down or drop-down menus provides a wide range of menu architecture that will meet most application needs.

A Menu event uses the standard OutTEGLTextXY and DefineMouseClickArea procedures to list and to create additional mouse click areas on the screen.

Even though the menu unit is comprehensive, TEGL Windows is not limited to a standard architecture of menus. The menu unit may be used as an example in creating other types of menu events; such as hanging menus which are not dependent on a bar type selection; or an icon menu, that when clicked on explodes to display a box full of icons that can be selected from.

The entries for the menu unit are created and linked at run-time. The entries may be manipulated, copied, or deleted as required within the program. In comparison, some systems offer a external menu compiler which links the menu with the program at compile time. The advantages to an external menu compiler are minimal, and it adds another step in creating a menu system.

The advantages to creating dynamic menus at run-time, is the ability to create a menu system that is based on an external text file (ie. the menu text selections may be stored in a text file and read in at run-time to create a menu).

Creating a Menu

Creating a bar menu is a two step process. The first is to create the entry text list that is associated with a option menu. The second is the creation of the menu bar from which option menus may be selected. You may use the first step by itself to attach an Option Entry list to icon, instead of a bar.

Creating a entry text list

An entry text list is simply an linked chain of text entries, with a root entry for each text list.

```
+------------+      +-----------+--+       +-----------+--+
|AnchorOMPtr|----->|OptionMenu |01|------>|OptionMenu |02|----->nil
+------------+      +-----+-----+--+       +-----+-----+--+
                          *                      *
                    +-----------+--+       +-----------+--+
                    |OptionEntry|01|       |OptionEntry|01|
                    +-----+-----+--+       +-----+-----+--+
```

```
                         *                      *
                   +-----------+--+      +-----------+--+
                   |OptionEntry|02|      |OptionEntry|02|
                   +-----+-----+--+      +-----+-----+--+
                         *                      *
                   +-----------+--+            nil
                   |OptionEntry|03|
                   +-----+-----+--+
                         *
                        nil

        +--------------------------------------------------+
        |  OptionMenu  = record                            |
        |                   NextOM          : OptionMPtr;+ |
        |                   numofentries    : word;      | |
------->|                   maxwidth        : word;      +--|----------->
        |                   padding         : word;        |
        |                   fonttype        : pointer;     |
        |            +----- FirstEntry      : OptionEPtr;  |
        |            |      CurrentEntry    : OptionEPtr;  |
        |            |   end;                              |
        |            +------------+                        |
        +-------------------------|------------------------+
                                  *
        +-----------------------------------------------+--+
        |  OptionEntry = record                         |01|
        |            +----- NextOE          : OptionEPtr;+--
        |            |      entryline       : string[40];  |
        |            |      entryactive     : boolean;     |
        |            |      entrycolor      : integer;     |
        |            |      entrycallproc   : callproc;    |
        |            |   end;                              |
        |            +------------+                        |
        +-------------------------|------------------------+
                                  *
```

OM is a short form for an OptionMenu record. This is the header or the root entry for an entry list. The header contains information regarding the number of entries, the maximum width of the entries, the amount of padding on left and right when displayed and the font type that is used. By duplicating the header with a different set of parameters, an Option Entry list may be chained to two or more headers to allow for different fonts.

```
              +-----------+      +-----------+
         ---->|OptionMenu |----->|OptionMenu |--->
              +-----+-----+      +-----+-----+
                    |                  |
                    |------------------+
                    |
                    *
              +-----------+
```

```
                    |OptionEntry|
                    +-----+-----+
                          *
```

OE is a short form for an OptionEntry record. There is no limit to the number of OE records that a list can contain, with the exception that the number of entries cannot be greater than the size of the screen when the OE list is displayed. This is a limitation of the ListOptionMenu procedure within the Menu unit and the screen vertical size, rather then a maximum entry limitation. The ListOptionMenu event could be modified to accommodate lists greater then the screen size by displaying a portion of a list and adding another event to display the remainder.

The OE record contains the entry (text) line, as well as information on whether the entry line is active or inactive (place holder), its color, and the event that is called when it is selected.

CreateOptionMenu Function TEGLMENU

Function

Creates an Option Menu header.

Declaration

CreateOptionMenu(Fonttype:pointer): OptionMPtr;

Result type

Returns an Option Menu pointer type.

Remarks

Fonttype is one of the fonts in the font library.

The option menu header is used to build and reference the Option Entry list. Use this OM pointer when calling the procedure DefineOptions.

Restrictions

To create multiple OM headers with different fonts on a single OE list, use CreateShadowOM to automatically create and link the OE list to a new OM header.

See also

DefineOptions, CreateShadowOM

Example

```
var OM1, OM2 : optionmptr;

  OM1 := CreateOptionMenu(@font14);
```

```
OM2 := CreateOptionMenu(@script);
```

DefineOptions Procedure TEGLMENU

Function

Adds Option Entries to an Option Menu.

Declaration

```
DefineOptions(var OM; EntryStr:string; Active: boolean;
  p : callproc);
```

Remarks

The OM pointer must be defined by CreateOptionMenu before Option Entries may be added.

EntryStr is the text string to be displayed when the Option menu is opened. The EntryStr has two types of control character which may be embedded as part of the string. The q - is used to display a dotted separator line between options. To underline a character or a series of characters, add the value of 128 to the ascii value. The underline character is only valid for characters that do not have descenders.

Active specifies whether this entry is active (can be selected) or not active. Inactive entries are displayed as jagged characters.

p defines the Event that is associated with this menu entry. The p is attached automatically to the option entry when the option menu is displayed.

Restrictions

There are no limitations on the number of entries that can be defined under a single OM header. However, too many entries will list past the bottom of the screen.

See also

CreateOptionMenu, CreateShadowOM, UnderLineChar

Example

```
var OM1 : optionmptr;

OM1 := CreateOptionMenu(@font14);
DefineOptions(OM1,'DeskTop Info...',true,InfoOption);
DefineOptions(OM1,'--',false,nilunitproc);
DefineOptions(OM1,'Calculator',true,nilunitproc);
DefineOptions(OM1,'Clock',true,nilunitproc);
DefineOptions(OM1,'Snapshot',true,nilunitproc);
```

CreateShadowOM Function — TEGLMENU

Function

Creates a duplicate Option Menu Header with a different Font type.

Declaration

CreateShadowOM(OM:OptionMPtr; Fonttype:pointer) : OptionMPtr;

Result type

Returns an new Option Menu pointer type.

Remarks

OM must be an existing OptionMenu pointer.
Fonttype is one of the fonts in the font library.

Restrictions

The original OM pointer must be defined by CreateOptionMenu before a duplicate Option Menu header may be created.

See also

CreateOptionMenu, ResizeOptionMenu

Example

```
var OM1,OM2 : optionmptr;

OM1 := CreateOptionMenu(@font14);
DefineOptions(OM1,'DeskTop Info...',true,InfoOption);
DefineOptions(OM1,'--',false,nilunitproc);
DefineOptions(OM1,'Calculator',true,nilunitproc);
DefineOptions(OM1,'Clock',true,nilunitproc);
DefineOptions(OM1,'Snapshot',true,nilunitproc);
```

```
OM2 := CreateShadowOM(OM1,@Script);
```

ResizeOptionMenu Procedure TEGLMENU

Function

Allows an Option Menu header to recalculate the size of the option menu window when changing the font type.

Declaration

ResizeOptionMenu(OM:OptionMPtr; Fonttype: pointer)

Remarks

OM must be an existing OptionMenu pointer.
Fonttype is one of the fonts in the font library.

See also

CreateOptionMenu, CreateShadowOM

Example

```
var OM1 : optionmptr;

OM1 := CreateOptionMenu(@font14);
DefineOptions(OM1,'DeskTop Info...',true,InfoOption);
DefineOptions(OM1,'--',false,nilunitproc);
DefineOptions(OM1,'Calculator',true,nilunitproc);
DefineOptions(OM1,'Clock',true,nilunitproc);
DefineOptions(OM1,'Icon Display',true,Icons);

ResizeOptionMenu(OM1,@Script);
{ -- Changes the font type @Font14 to @Script}
```

ToggleCheckMark Procedure TEGLMENU

Function

Changes the first character of an entry string to 0x30 (check mark) or a 0x32 (space).

Declaration

ToggleCheckMark(OMNum,OENum : word; status:boolean);

Remarks

OMNum is the position of the Option Menu header relative to the AnchorOMPtr. OENum is the position of the Option Entry relative to the OM header.

Status of True will change the first character of the entry to a checkmark, False will change the character to a space.

See also

ToggleEntryStatus, ReplaceOptionText

Example

```
var OM1 : optionmptr;

OM1 := CreateOptionMenu(@font14);
DefineOptions(OM1,'  Show as Icons ',true,ViewOptionToggle);
DefineOptions(OM1,'  Show as Text  ',true,ViewOptionToggle);
DefineOptions(OM1,'-',false,nilunitproc);
DefineOptions(OM1,'  Sort by Name  ',true,ViewOptionToggle);
DefineOptions(OM1,'  Sort by Date  ',true,ViewOptionToggle);
DefineOptions(OM1,'  Sort by Size  ',true,ViewOptionToggle);
DefineOptions(OM1,'  Sort by Type  ',true,ViewOptionToggle);

ToggleCheckMark(1,7,TRUE);
{puts a check mark at the front of Sort by Type}
```

ToggleEntryStatus Procedure TEGLMENU

Function

Sets an Option entry to active or not active.

Declaration

ToggleEntryStatus(OMNum,OENum:word; status: boolean)

Remarks

OMNum is the position of the Option Menu header relative to the AnchorOMPtr.

OENum is the position of the Option Entry relative to the OM header.

Status of True will set the entry as active, False will set the entry to nonactive. Active specifies whether this entry is active (can be selected) or nonactive. Nonactive entries are displayed as jagged characters.

See also

ToggleCheckMark, ReplaceOptionText

Example

```
var OM1 : optionmptr;

OM1 := CreateOptionMenu(@font14);
DefineOptions(OM1,'DeskTop Info...',true,InfoOption);
DefineOptions(OM1,'--',false,nilunitproc);
DefineOptions(OM1,'Calculator',true,nilunitproc);
DefineOptions(OM1,'Clock',true,nilunitproc);
DefineOptions(OM1,'Snapshot',true,Snapshot);

ToggleEntryStatus(1,5,FALSE); {toggles Snapshot off}
```

ReplaceOptionText Procedure TEGLMENU

Function

Replaces Option entry string by another text string.

Declaration

ReplaceOptionText(OMNum,OENum : word; EntryStr: string)

Remarks

OMNum is the position of the Option Menu header

relative to the AnchorOMPtr.

OENum is the position of the Option Entry relative to the OM header.

EntryStr is a replacement text string that will be displayed when the Option menu is opened. The EntryStr has two types of control character which may be embedded as part of the string. The q - is used to display a dotted separator line between options. To underline a character or a series of characters, add the value of 128 to the ascii value. The underline character only works with characters that do not have descenders.

See also

ToggleCheckMark, ToggleEntryStatus

Example

```
VAR OM1 : optionmptr;

  OM1 := CreateOptionMenu(@font14);
  DefineOptions(OM1,'DeskTop Info...',true,InfoOption);
  DefineOptions(OM1,'--',false,nilunitproc);
  DefineOptions(OM1,'Calculator',true,nilunitproc);
  DefineOptions(OM1,'Clock',true,nilunitproc);
  DefineOptions(OM1,'Icon Display',true,Icons);

  { -- Replaces "Icon Display" with "Text Display"}
  ReplaceOptionText(1,5,"Text Display");
```

ToggleOptionBar Procedure TEGLUNIT

Function

Inverts mouse click areas.

Declaration

```
ToggleOptionBar(ifs : ImageStkPtr;
  Opt,LastOpt: MsClickPtr);
```

Remarks

Opt and LastOpt mouse click areas are inverted. It is assumed that LastOpt has

already been inverted and this call would return it to normal.

SetOptionMenuColors Procedure TEGLMENU

Function

Changes the menu entry colors.

Declaration

SetOptionMenuColors(activecolor,inactivecolor:word);

Remarks

activecolor is the text color for active entries.

inactivecolor is the text color for entries that are currently inactive but have entry positions within the menu.

See also

SetOptionMenuBorderColor

Example

```
SetOptionMenuColors(Black,LightGray);
```

SetOptionMenuBorderColor Procedure TEGLMENU

Function

Changes the color of the option menu border.

Declaration

SetOptionMenuBorderColors(color:word)

Remarks

color is the color of the border.

See also

SetOptionMenuColors

Example

```
SetOptionMenuBorderColor(white);
```

SetHideSubMenu Procedure TEGLMENU

Function

Toggles the hiding of sub menus.

Declaration

SetHideSubMenu(OnOff : Boolean);

Remarks

Default is true. When a submenu is pulled down from a bar menu it is normally hidden when a selection is made. If set to false then the pulldown is left displayed until the selection that was made returns.

Example

```
SetHideSubMenu(True);
```

Creating a Bar Menu

A bar menu is one of the more popular methods of creating a user interface. As mentioned before, a bar menu is simply another event with the event handler set to BarOptionMenu. BarOptionMenu is activated whenever the mouse cursor passes by the one of the defined mouse click areas on the bar.

When BarOptionMenu is activated, OptionMenuSelection is called in place of the TEGLSupervisor.

There are three activities within a menu system that require a rewrite of the TEGLSupervisor. OptionMenuSelection checks if

The mouse is clicked outside the menu bar or menu window thus closing any active menus and returning back to the TEGL supervisor.

Sensing the mouse cursor movement to another bar entry, thus closing any active menu and opening another menu window.

Sensing the mouse cursor moving to another entry within a menu and highlighting the entry.

CreateBarMenu Procedure TEGLMENU

Function

Creates a Bar window frame.

Declaration

CreateBarMenu(x,y,ln:word)

Remarks

x, y is the position of the bar menu frame.

ln is the pixel length of the bar.

See also

OutBarOption

Example

```
  CreateBarMenu(0,0,GetMaxX);
```

OutBarOption Procedure TEGLMENU

Function

Attaches an option menu (list) to a displayed text string on the BAR.

Declaration

OutBarOption(EntryStr:string; OM:OptionMptr)

Remarks

EntryStr is the bar text header that is associated with the OM list.

OM is the Option Menu header returned from CreateOptionMenu.

See also

CreateBarMenu

Example

```
VAR OM1 : optionmptr;

  OM1 := CreateOptionMenu(@font14);
  DefineOptions(OM1,'  Show as Icons ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Show as Text  ',true,ViewOptionToggle);
  DefineOptions(OM1,'-',false,nilunitproc);
  DefineOptions(OM1,'  Sort by Name  ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Sort by Date  ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Sort by Size  ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Sort by Type  ',true,ViewOptionToggle);

  CreateBarMenu(0,0,639);
  OutBarOption(' Options ',OM1);
```

SetBarTextColor Procedure — TEGLMENU

Function

Changes the default text color on the bar.

Declaration

SetBarTextColor(color:word)

Remarks

color is the default text color on the bar.

See also

SetBarMenuColor, SetBarBorderColor

Example

```
  SetBarTextColor(green);
```

SetBarMenuColor Procedure — TEGLMENU

Function

Changes the bar color.

Declaration

SetBarMenuColor(color:word)

Remarks

color is the default color for the bar.

See also

SetBarMenuColor, SetBarBorderColor

Example

```
SetBarMenuColor(blue);
```

SetBarBorderColor Procedure TEGLMENU

Function

Changes the bar border color and toggles the border on.

Declaration

SetBarBorderColor(color:word)

Remarks

color is the default border color for the bar.

See also

SetBarTextColor, SetBarBorderOff

Example

```
SetBarBorderColor(green);
```

SetBarBorderOff Procedure TEGLMENU

Function

Toggles the bar border off.

Declaration

SetBarBorderOff

Remarks

SetBarBorderColor resets the border on.

See also

SetBarBorderColor, SetBarTextColor

Example

```
SetBarBorderOff;
```

SetBarShadowtext Procedure TEGLMENU

Function

Toggles Bar Shadow Text on/off.

Declaration

SetBarShadowtext(OnOff:boolean)

Remarks

OnOff is a boolean type, where TRUE is on and FALSE is off.

Example

```
SetBarShadowText(True);
```

SetBarFillStyle Procedure TEGLMENU

Function

Sets the Bar Fill Style.

Declaration

SetBarFillStyle(pattern:word)

Remarks

Sets the pattern for the bar. The fill patterns are defined by constants in the Graph unit.

Pattern is a numeric type.

See also

SetFillStyle (Graph Unit).

Example

```
SetBarFillStyle(BkSlashFill);
```

SetBarMenuMargin Procedure TEGLMENU

Function

Sets the left margin on the barmenu.

Declaration

```
SetBarMenuMargin(Margin: Word);
```

Remarks

Margin is the desired left margin where the menu selections start at. This value is in pixels and the default is 16.

Can be used if a icon or some symbol should be displayed at the extreme left of the menu.

Example

```
  SetBarMenuMargin(32);
```

Icon Option Menus

Optionally you can attach a menu to an icon or an area of a frame.

The following procedure adds a drop down menu to any frame area.

DefineOptionClickArea Procedure TEGLMENU

Function

Attaches an option menu (list) to a frame or icon area.

Declaration

```
DefineOptionClickArea(var ifs; x,y,x1,y1:word; OM:OptionMPtr;
  Sense:boolean; OMType:byte)
```

Remarks

ifs is any ImageStkPtr. The x, y, x1, y1 are coordinates relative to a frame. This means that the upper left corner of a frame is considered 0,0.

OM is the Option Menu header returned from CreateOptionMenu.

Sense is either MSSense or MSClick. MSSense activates the menu event handler whenever the mouse cursor passes over the defined mouse click areas.4MSClick requires the right mouse button to be pressed while the mouse cursor is on the mouse click area.

OMType is the enumerated type of UpperRight, UpperLeft, LowerRight, and LowerLeft, which specifies whether the menu pop-down at the upper right or upper left corner, or pop-up at the lower right or lower left corner.

See also

DefineMouseClickArea, ResetOptionMenuEvents

Example

```
VAR OM1 : optionmptr;

  OM1 := CreateOptionMenu(@font14);
  DefineOptions(OM1,'  Show as Icons ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Show as Text  ',true,ViewOptionToggle);
  DefineOptions(OM1,'-',false,nilunitproc);
  DefineOptions(OM1,'  Sort by Name  ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Sort by Date  ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Sort by Size  ',true,ViewOptionToggle);
  DefineOptions(OM1,'  Sort by Type  ',true,ViewOptionToggle);

  PushImage(530,320,624,340);
  PutPict(530,320,@ImageCredits,black);
  DefineOptionClickArea(StackPtr,0,0,93,19,OM1,MSClick,
    LowerRight);
```

ResetOptionMenuEvents Procedure TEGLMENU

Function

Eliminates duplicate menu events where the frame has been closed.

Declaration

ResetOptionMenuEvents

Remarks

The Menu unit keeps track of menu to frame attachments. In most cases the attachment is permanent, that is, until the program terminates. However in some cases, like the icon editor, the menu to frame attachment changes every time the icon editor explodes or implodes an icon image. Since the Menu unit has no way of knowing whether the attachment still exists, a special procedure was created to eliminate duplicate or nonexistent event relationships.

The only problem with not calling ResetOptionMenuEvents would be an accumulation of menu events for non-existing frames. Eventually the heap area will overflow.

See also

DefineOptionClickArea

Example

```
VAR OM1 : Optionmptr;

OM1 := CreateOptionMenu(@font14);
DefineOptions(OM1,'  Show as Icons ',true,ViewOptionToggle);
DefineOptions(OM1,'  Show as Text  ',true,ViewOptionToggle);
DefineOptions(OM1,'-',false,nilunitproc);
DefineOptions(OM1,'  Sort by Name  ',true,ViewOptionToggle);
DefineOptions(OM1,'  Sort by Date  ',true,ViewOptionToggle);
DefineOptions(OM1,'  Sort by Size  ',true,ViewOptionToggle);
DefineOptions(OM1,'  Sort by Type  ',true,ViewOptionToggle);

PushImage(530,320,624,340);
PutPict(530,320,@imagecredits,black);
DefineOptionClickArea(StackPtr,0,0,93,19,OM1,MSClick,LowerRight);
PopImage;

PushImage(530,320,624,340);
PutPict(530,320,@imagecredits,black);
DefineOptionClickArea(stackptr,0,0,93,19,OM1,MSClick,LowerRight);
ResetOptionMenuEvents;
```

Chapter 6 - Mouse, Keyboard and Timer Handlers

Interrupt Handlers (TEGLIntr)

The mouse is perhaps one of the most outlandish devices ever conceived as an interface for computer system (at least in programming it). However, in the world of GUI, the mouse is a mandatory device.

Programming for a mouse is a programmer's nightmare, simply because it adds another level of interfacing. Conceptually, keyboard and mice do not mix. As an example, the mouse is dependent on screen location and whether the user had clicked the mouse at a specific location on the screen and whether that location was on an icon. The keyboard, on the other hand, is almost a direct path between pressing a key and executing a subroutine (ie. if keypress then do something).

The programmer is required to write two separate routines for the same function to handle this mix of interfaces. As well, some systems do not have a mouse, so you cannot rely on the mouse pointer being available on all systems.

TEGL Windows Toolkit, of course, provides an almost seamless integration of the two devices. On systems without a mouse, TEGL will emulate the mouse by using the cursor keys on the numeric keypad. On systems with a mouse, the cursor keys may be used simultaneously to move the mouse cursor around. A key may also be attached to an icon/event, having the same effect as the mouse clicking on the icon.

Interrupts

The TEGLIntr unit is comprised of four captured interrupts: The keyboard interrupt (int $09), the mouse subroutine interrupt (function 12), the timer interrupt (int $08) and the control break handler (int $1B).

SwapTEGLIntrOff and SwapTEGLIntrON should be called just before and after a call to Exec to restore and then to recapture interrupt vectors.

SwapTEGLIntrOff Procedure — TEGLINTR

Function

Restores all interrupts to the original saved vectors.

Declaration

SwapTEGLIntrOff

Remarks

All interrupts are initially turned on.

See also

SwapTEGLIntrOn

SwapTEGLIntrOn Procedure TEGLINTR

Function

Saves and initialize the required TEGL interrupts.

Declaration

SwapTEGLIntrOn

Restrictions

SwapTEGLIntrOn cannot be called more then once in succession, otherwise the system will hang.

See also

SwapTEGLIntrOn

Mouse Emulation

The mouse cursor is an internal function of the TEGL mouse unit, rather than using the cursor provided by the mouse driver. This way a mouse cursor is always available even on systems that do not have a mouse.

The support for the emulated mouse is identical, in all respects, to the actual mouse driver.

In order to provide a seamless integration of the mouse and keyboard, the Mouse function 12 interrupt $33 is used to capture the mouse hardware interrupts, and keyboard interrupt $09 is used to capture key codes. Since both are hardware interrupts, a KBMouseBusy flag is used to serialize any conflict if both interrupts occurs at the same time.

The emulated mouse cursor is controled by the following primitives. They may be used ONLY if the MouseShow flag is false, otherwise you may find mouse droppings on the screen.

MCursorOff Procedure TEGLINTR

Function

Switches the Emulated Mouse Cursor off.

Declaration

MCursorOff

Restrictions

Use ONLY when MouseShow flag is False.

See also

MCursorOn, MSetPos

MCursorOn Procedure TEGLINTR

Function

Switches the Emulated Mouse Cursor on.

Declaration

MCursorOn(Xpos,Ypos : Word);

Remarks

Xpos, Ypos is the relative screen coordinates from the upper left corner of 0,0.

Restrictions

Use ONLY when MouseShow flag is False.

See also

MCursorOff, MSetPos

MSetPos Procedure TEGLINTR

Function

Sets a new position for the Emulated Mouse Cursor.

Declaration

MSetPos(XPos, YPos: Word);

Remarks

xpos, ypos is the relative screen coordinates from the upper left corner of 0,0.

Restrictions

The emulated mouse cursor must be on before setting a new position.

Use ONLY when MouseShow flag is False.

See also

MCursorOff, MCursorOn

Standard Mouse Functions

ShowMouse Procedure TEGLINTR

Function

Display a mouse cursor at current Mouse_Xcoord, Mouse_Ycoord.

Declaration

ShowMouse;

See also

HideMouse, SetMousePosition, CursorShape

HideMouse Procedure TEGLINTR

Function

Hides mouse cursor.

Declaration

HideMouse

See also

ShowMouse, SetMousePosition, CursorShape

SetMousePosition Procedure TEGLINTR

Function

Sets x,y coordinates of mouse cursor.

Declaration

SetMousePosition(MouseX,MouseY : word)

Remarks

MouseX, MouseY are relative coordinates from the upper left corner of the screen 0,0.

See also

ShowMouse, HideMouse, CursorShape

CursorShape Procedure TEGLINTR

Function

Sets the mouse cursor shape.

Declaration

CursorShape(Shape:Masktype)

Remarks

Sets the mouse cursor shape to the bit pattern specified in Shape.

Masktype is predefined as follows:

```
type
    MaskType = array[0..1, 0..15] of word;
```

The mouse shape is based on the underlying byte values contained in the Shape array. The Shape array is 64 bytes long, with the first 32 bytes corresponding to a 16 by 16 screen mask, and the remaining 32 bytes corresponding to a 16 by 16 cursor mask. The first 32 bytes are ANDed to the screen, followed by ORing the second 32 bytes with the screen pixels to create the final mouse image.

For example the PointingHand Masktype is defined as a constant as follows:

```
PointingHand: MaskType =
  (($E1FF,$E1FF,$E1FF,$E1FF,$E1ff,$E000,$E000,$e000,   { Screen Mask }
    $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000),

   ($1E00,$1200,$1200,$1200,$1200,$13ff,$1249,$1249,   { Cursor Mask }
    $1249,$9001,$9001,$9001,$8001,$8001,$8001,$FFFF));
```

The resulting type is:

```
Screen Mask

1110000111111111     =  $E1FF
1110000111111111     =  $E1FF
1110000111111111     =  $E1FF
1110000111111111     =  $E1FF
1110000111111111     =  $E1FF
1110000000000000     =  $E100
1110000000000000     =  $E100
1110000000000000     =  $E100
```

```
0000000000000000     =  $0000
0000000000000000     =  $0000
0000000000000000     =  $0000
0000000000000000     =  $0000
0000000000000000     =  $0000
0000000000000000     =  $0000
0000000000000000     =  $0000
0000000000000000     =  $0000
```

Cursor Mask

```
0001111000000000     =  $1E00
0001001000000000     =  $1200
0001001000000000     =  $1200
0001001000000000     =  $1200
0001001000000000     =  $1200
0001001111111111     =  $13FF
0001001001001001     =  $1249
0001001001001001     =  $1249
0001001001001001     =  $1249
1001000000000001     =  $9001
1001000000000001     =  $9001
1000000000000001     =  $8001
1000000000000001     =  $8001
1000000000000001     =  $8001
1000000000000001     =  $8001
1111111111111111     =  $FFFF
```

There are 5 masktype constants defined in the TEGLIntr unit. They are: Pointing Hand, HourGlass, Standard, DiagCross, and CheckMark.

See also

ShowMouse, HideMouse, SetMouseHotSpot

SetMouseHotSpot Procedure — TEGLINTR

Function

Sets the cursor hot-spot values relative to the upper-left corner of the mouse cursor image.

Declaration

SetMouseHotSpot(x,y : word)

Remarks

x, y are relative coordinates from the upper left corner of the mouse cursor image 0,0.

See also

CursorShape

SetMouseColor Procedure TEGLINTR

Function

Sets the mouse cursor color.

Declaration

SetMouseColor(Color:word)

Remarks

Sets the current Mouse Cursor Color to Color. Available colors are defined in the Graph Unit.

See also

CursorShape

MousePosition function TEGLINTR

Function

Gets the Mouse Cursor coordinates and button information.

Declaration

MousePosition(VAR MouseX,MouseY : Word) : Word;

Result type

Returns the mouse button status. Left button - 1, Right button - 2, both buttons - 3.

Remarks

MouseX,MouseY are relative coordinates from the upper left corner of the screen (0,0).

This function is no longer required in version 2.00, since the the information above are provided in the global variables Mouse_XCoord, Mouse_YCoord and Mouse_Buttons respectively.

See also

GetButtonReleaseInfo, GetButtonPressInfo, ClearButtonInfo

Example

```
VAR mp,x,y : Word;

  mp := MousePosition(x,y);
  IF Integer(mp) = -3 THEN   { -- Both buttons down }
    BEGIN
    END;
```

GetButtonReleaseInfo Procedure TEGLINTR

Function

Gets the Mouse Cursor button release information.

Declaration

GetButtonReleaseInfo(Button:word; VAR ButtonStat, ButtonRelease,Xpos,Ypos:word)

Remarks

Button specifies for which button information is required.

ButtonStat is the current button status information.

ButtonRelease is the number of times the button has been released.

Xpos, Ypos specifies the coordinates where the button was last released.

The information is reset back to zero after the information has been read.

See also

MousePosition, GetButtonPressInfo, ClearButtonInfo

GetButtonPressInfo Procedure TEGLINTR

Function

Gets the Mouse Cursor button press information.

Declaration

```
GetButtonPressInfo(Button : word; VAR ButtonStat,
  ButtonRelease,Xpos,Ypos:word)
```

Remarks

Button specifies for which button information is required.

ButtonStat is the current button status information.

ButtonPress is the number of times the button has been pressed.

Xpos,Ypos specifies the coordinates where the button was last pressed.

The information is reset back to zero after the information has been read.

See also

MousePosition, GetButtonReleaseInfo, ClearButtonInfo

ClearButtonInfo Procedure — TEGLINTR

Function

Clears the Mouse button info counters.

Declaration

```
ClearButtonInfo;
```

See also

GetButtonReleaseInfo, GetButtonPressInfo

SetMouseMinMax Procedure — TEGLINTR

Function

Sets the Mouse Cursor minimum and maximum coordinates.

Declaration

```
SetMouseMinMax(MinX,MinY,MaxX,MaxY:word)
```

Remarks

MinX, MinY are the minimum relative coordinates that the mouse may travel. MaxX, MaxY are the maximum relative coordinates that the mouse may travel.

See also

SetMousePosition

FrozenMouse Procedure TEGLINTR

Function

Prevents the mouse from moving when updating the screen.

Declaration

FrozenMouse

Remarks

Certain EGA registers cannot be read reliably. Rather then attempting to read and restore the register with each movement of the mouse, it is more economical to simply freeze the mouse, while the screen is being updated.

FrozenMouse retains a counter on the number of times the mouse is frozen. In order to unfreeze the mouse, the same number of UnFreeze calls must be made.

Restrictions

FrozenMouse may be used if the screen update is temporary (ie. XorBox), or the second EGA page is being updated. Care must be taken that the mouse cursor is not overlapping the updated area, otherwise mouse droppings may result.

See also

FreezeMouse, UnFreezeMouse

FreezeMouse function TEGLINTR

Function

Prevents the mouse from moving or being overwritten when updating the screen.

Declaration

FreezeMouse(x,y,x1,y1:word)

Result type

Returns the last MouseShow status.

Remarks

Certain EGA registers cannot be read reliably. Rather then attempting to read and restore the register with each movement of the mouse, it is more economical to simply freeze the mouse, while the screen is being updated.

FreezeMouse differs from FrozenMouse in that a check is made on whether the mouse cursor overlaps the updated area. If the mouse cursor overlaps the update area, the mouse is hidden until UnFreeze displays the mouse.

FreezeMouse also retains a counter on the number of times the mouse is frozen. In order to unfreeze the mouse, the same number of UnFreeze calls must be made.

Restrictions

FrozenMouse may be used if the screen update is temporary (ie. XorBox), or if the second EGA video page is being updated.

See also

FrozenMouse, UnFreezeMouse

UnFreezeMouse Procedure — TEGLINTR

Function

Releases the mouse from a frozen or freeze status.

Declaration

UnFreezeMouse(Mshow:boolean)

Remarks

Mshow is the mouse show status returned from FreezeMouse, or use the global MouseShow flag if FrozenMouse was called.

FreezeMouse and FrozenMouse retain a counter on the number of times the mouse is frozen. In order to unfreeze the mouse, the same number of UnFreeze calls must be made.

See also

FrozenMouse, FreezeMouse

SetMouseSensitivity Procedure — TEGLINTR

Function

Sets the mouse-to-cursor movement sensitivity.

Declaration

```
SetMouseSensitivity(Xsense,Ysense,Threshold:
word)
```

Remarks

Xsense defines the horizontal movement sensitivity.

Ysense defines the vertical movement sensitivity.

The sensitivity numbers range from 1 through 100, where 50 specifies the default mickey factor of 1. The mouse-to-cursor movement is more sensitive at higher numbers.

The threshold parameter sets the ratio at which the mouse-to-cursor movement is doubled. This range of this parameter is also 1 through 100. The lower the threshold, the more sensitive the mouse.

See also

GetMouseSensitivity

GetMouseSensitivity Procedure — TEGLINTR

Function

Returns the mouse-to-cursor movement sensitivity scaling factors previously set by SetMouseSensitivty.

Declaration

```
GetMouseSensitivity(VAR Xsense,Ysense,
  Threshold:word)
```

Remarks

Xsense defines the horizontal movement sensitivity.

Ysense defines the vertical movement sensitivity.

The sensitivity numbers range from 1 through 100, where 50 specifies the default mickey factor of 1. The mouse-to-cursor movement is more sensitive at higher

numbers.

The threshold parameter is the ratio at which the mouse-to-cursor movement is doubled. This range of this parameter is also 1 through 100. The lower the threshold, the more sensitive the mouse.

See also

SetMouseSensitivity

SetKeyBoardMouse Procedure — TEGLINTR

Function

Toggles the keyboard mouse on or off.

Declaration

SetKeyBoardMouse(ON_OFF : boolean)

Remarks

The cursor keys leftarrow downarrow uparrow rightarrow, on the keyboard, may be used to emulate the mouse movements. SetKeyBoardMouse(FALSE) will turn off the emulation, to allow GetCh to retrieve the keycode.

Restrictions

SetKeyBoardMouse will have no effect on TEGL's keyboard events, (ie. the cursor keys may be assigned functions by means of AddCaptureKey), which will have priority over the keyboard mouse.

See also

SetKBSteps, GetKBSteps

SetKBSteps Procedure — TEGLINTR

Function

Sets the amount of pixel movement with each cursor key press.

Declaration

SetKBSteps(xsteps,ysteps,sfxsteps,sfysteps: word)

Remarks

xsteps, ysteps are the positive incremental values

for moving the mouse cursor to the next position. Initial values are (x=12,y=8).

sfxsteps, sfysteps are the positive incremental value for moving the mouse cursor to the next position when using the shiftkey in conjunction with the leftarrow downarrow uparrow rightarrow keys. Initial values are (x=2,y=1).

Restrictions

SetKBSteps will have no effect on TEGL's keyboard events, (ie. the cursor keys may be assigned functions by means of AddCaptureKey), which will have priority over the keyboard mouse.

See also

SetKeyBoardMouse, GetKBSteps

GetKBSteps Procedure — TEGLINTR

Function

Returns the pixel movement value set for the keyboard mouse.

Declaration

GetKBSteps(xsteps,ysteps,sfxsteps,sfysteps: word)

Remarks

xsteps, ysteps are the positive horizontal and vertical step increments.

sfxsteps, sfysteps are the positive horizontal and vertical step increments when using the shiftkey in conjunction with the leftarrow downarrow uparrow rightarrow keys.

See also

SetKeyBoardMouse, SetKBSteps

Timer Functions

A timer tick has the standard resolution of interrupting any process within the system, 18 times a second. TEGL Windows uses the captured timer interrupt to decrement counters and set a flag when the counter is zero. TEGLSupervisor monitors the status of the flag and calls the attached event when the flag is set. Thus timed events are processed outside the critical timer tick interrupt.

Timer events may be used as clocks, background tasks, print spoolers etc.

SwapTimerOut Procedure TEGLINTR

Function

Restores the original timer vectors.

Declaration

SwapTimerOut

Remarks

Use SwapTimerOut if you need to turn the timer off.

See also

SwapTimerIn

SwapTimerIn Procedure TEGLINTR

Function

Captures the original timer vectors and sets the interrupt vectors to point at TEGL's timer function.

Declaration

SwapTimerIn

Remarks

The timer interrupt is originally swapped in.

Restrictions

SwapTimerIn cannot be called more then once in succession, otherwise the system will hang.

See also

SwapTimerIn

SetTimerStart Procedure TEGLINTR

Function

Sets the timer value of timepiece counter.

Declaration

```
SetTimerStart(VAR Timepiece : TimeRecPtr;
  Timeset: Word)
```

Remarks

Timepiece is of the type TimeRecPtr. If Timepiece is set to Nil, a timepiece record is created and initialized to timeset.

Timeset is a word value counter. A value of 18 is equivalent of 1 second.

See also

ResetTimerFlag

ResetTimerFlag Procedure — TEGLINTR

Function

Resets the flag that indicates the completion of a cycle. A cycle is when the counter reaches zero and is reset back to its original value.

Declaration

```
ResetTimerFlag(Timepiece:TimeRecPtr)
```

Remarks

Timepiece is of the type TimeRecPtr.
timepiece is created by SetTimerStart.

See also

SetTimerStart

DropTimerCount Procedure — TEGLINTR

Function

Deletes a timepiece record from the timer event chain.

Declaration

```
DropTimerCount(Timepiece : TimeRecPtr)
```

Remarks

Timepiece is of the type TimeRecPtr.
Timepiece is created by SetTimerStart.

See also

SetTimerStart

TimerSwtich Procedure TEGLINTR

Function

Toggles the timer handler on or off.

Declaration

TimerSwitch(onoff:boolean)

Remarks

onoff sets the status on whether the timer event chain is scanned and decremented. A boolean value of FALSE stops the counters from being decremented. A boolean value of TRUE resets the counters back to their original values and causes the counters within the timer event chain to be decremented 18 times a second.

TimerSwitch does not remove the timer interrupt vectors.

See also

SwapTimerOut, SwapTimerIn

Keyboard Interrupt Events

There are two levels at which the keyboard interrupt may be used. At the higher Keyboard Event level (monitored by the TEGLSupervisor), complete events, like swapping rotating windows, may be attached to a key on the keyboard. However, at the lower level setting the keycall parameter in AddCaptureKey to point at a key handler allows low level functions like positioning the mouse cursor to be performed.

A good example of a key handler is the default mouse click handler. The enterkey is used to automatically position the mouse cursor on the first defined mouse click area and simulates the holding down of the mouse right button, until the key is released.

The higher Keyboard Event level is set with a call to DefineLocalKeyClickArea and DefineGlobalKeyClickArea within TEGLUnit. The keycall parameter in AddCaptureKey is set to NilKeyCallProc. Instead of calling an external callproc, the keys are stacked in a keyboard buffer that is monitored by the TEGLSupervisor.

This TEGL keyboard buffer is separate from the normal keyboard buffer. The TEGLKeyPressed and TEGLReadKey functions are provided to check and read captured keys.

Note: The keyboard handler uses scan codes rather then translated Ascii

codes.

Keyboard Scan Codes

$01 esckey
$02 key 1key !
$03 key 2key @
$04 key 3key #
$05 key 4key $
$06 key 5key %
$07 key 6key ^
$08 key 7key &
$09 key 8key *
$0A key 9key (
$0B key 0key)
$0C {key -} _
$0D key =key +
$0E backspace
$0F forwtabbacktab
$10 key Q
$11 key W
$12 key E
$13 key R
$14 key T
$15 key Y
$16 key U
$17 key I
$18 key O
$19 key P
$1A [{
$1B] }
$1C enterkey
$1D ctrlkey
$1E key A
$1F key S
_
$20 key D
$21 key F
$22 key G
$23 key H
$24 key J
$25 key K
$26 key L
$27 ; :
$28 ' "
$29 ` ~
$2A shiftkey Left
$2B {key }
$2C key Z
$2D key X
$2E key C
$2F key V
$30 key B
$31 key N
$32 key M
$33 key ,key <
$34 key .key >
$35 key /key ?
$36 shiftkey Right
$37 prtsckeykey *
$38 altkey
$39 {spacebar}
$3A {capslock}
$3B key F1
$3C key F2
$3D key F3
$3E key F4
$3F key F5
$40 key F6
$41 key F7
$42 key F8
$43 key F9
$44 f10
$45 numlock
$46 scrlock
$47 homekey key 7
$48 uparrow key 8
$49 pgupkey key 9
$4A key -
$4B {leftarrow} {key 4}
$4C key 5
$4D rightarrow key 6
$4E key +
$4F endkey key 1
$50 downarrow key 2
$51 pgdnkey key 3
$52 inskey key 0
$53 delkey key .
$54 sysreq
$85 bigfrontF11keyback
$86 bigfrontF12keyback

AddCaptureKey Procedure TEGLINTR

Function

Adds a keyboard scancode to the keyboard handler for capturing, or for processing immediately when the key is pressed.

Declaration

AddCaptureKey(Keycode:word;Repeatkey:Koolean;

  Keycall:keybrdcallproc)

Remarks

Keycode is the scan code of the keys on the keyboard. This is different from the ascii code that is usually translated and passed by DOS. Use the scancode value listed in the scancode table.

Repeatkey is set to TRUE if the key is expected to repeat. False if the key must be released before generating another interrupt.

Keycall is the key call procedure when the keyboard handler captures the key. If keycall is set to NilKeyCallProc the scancode of the capture key is added to the TEGL keyboard buffer.

AddCaptureKey can stack the same scan code any number of times, however, only the most recent entry in the Scancode chain is used.

See also

DeleteCaptureKey

DeleteCaptureKey Procedure TEGLINTR

Function

Removes a keyboard scancode from the keyboard scancode chain.

Declaration

DeleteCaptureKey(Keycode : Word)

Remarks

Keycode is the scan code of the keys on the keyboard. This is different from the ascii code that is usually translated and passed by DOS.

If the same scan code is stacked more then once the most recent entry in the Scancode chain is deleted.

See also

AddCaptureKey

TEGLReadkey Function TEGLINTR

Function

Reads a scan code from the TEGL keyboard buffer.

Declaration

TEGLReadkey

Result type

Returns the first captured scan code in the TEGL keyboard buffer.

Restrictions

Use TEGLKeyPressed to check if any scan codes are in the TEGL keyboard buffer.

See also

TEGLKeyPressed

TEGLKeyPressed Function — TEGLINTR

Function

Returns True if a scan code is captured; False otherwise.

Declaration

TEGLKeyPressed

Result type

Boolean

Remarks

The scan code is added to the TEGL keyboard buffer.

See also

TEGLReadKey

NilKeyCallProc Function — TEGLINTR

Function

Dummy function to use a place holder.

Declaration

NillKeyCallProc

Returns

Boolean.

Remarks

This function always returns false.

See also

AddCaptureKey.

Keyboard Miscellaneous

SetShiftKeys Procedure — TEGLINTR

Function

Toggles the Shift flags on/off.

Declaration

SetShiftKeys(ShiftFlag:byte; OnOff:boolean)

Remarks

Shiftflag may be one of the types as follows:

```
TYPE
    Sk_RightShift  = $01;
    Sk_LeftShift   = $02;
    Sk_CtrlShift   = $04;
    Sk_AltShift    = $08;
    Sk_ScrollLock  = $10;
    Sk_NumLock     = $20;
    Sk_CapsLock    = $40;
    Sk_InsLock     = $80;
```

OnOff sets the above bits to on True or off False.

Show Button Status

The TEGL.PAS demonstration program uses the DEBUGUNT.PAS unit to display the mouse button status through a menu selection.

ShowButtonStatus Event — FONTTEST

Function

An Event that displays the mouse button status.

Remarks

Information is displayed on the number of times the mouse buttons have been pressed and released. Shows the last coordinates where the mouse button was pressed and

the coordinates where the mouse button was released.

Chapter 7 - Assembly Language Graphics

Assembler Graphics

The FASTGRPH unit is the engine that provides the speed that is seen in the TEGL Windows Toolkit. Most of the graphics tools are written in assembler, with some of the noncritical support routines written in Pascal.

Between the FASTGRPH and TGRAPH units programs can be made that require no other graphic support.

Graphics primitives are accessed through procedural pointers. When a graphics mode is selected (EGA640x350x16 etc...) the pointers are initialized point at the correct support routines. Graphics primitives cannot be called before a graphics mode is selected. If they are called then the program will probably crash severely and a reset may be required.

Setting Video Modes

The following Types and Consts relate to detecting and selecting video modes.

The VidID type is passed as a parameter to VideoID to determine the graphics equipment available.

```
VidID = RECORD
  Video0Type   : Byte;
  Display0Type : Byte;
  Video1Type   : Byte;
  Display1Type : Byte;
END;
```

The graphics adaptor card detected is returned in the Video0Type field. Here are a list of the Constants and values and whether they are currently supported by the toolkit.

```
TG_None   = $00;    no graphics adaptor
TG_MDA    = $01;    monochome display, not supported
TG_CGA    = $02;    Color graphics, supported
TG_EGA    = $03;    Enhanced graphics, supported
TG_MCGA   = $04;    Multicolor graphics array, not supported
TG_VGA    = $05;    Video graphics array, not supported
TG_HGC    = $80;    Hercules graphics, supported
TG_HGCPlus= $81;    Hercules plus, not supported
TG_InColor= $82;    Hercules incolor, no supported
```

CGA640x200x2 Procedure — FASTGRPH

Function
Sets the video mode to 640 x 200 in 2 colors.

Declaration
CGA640x200x2

Remarks
This procedure switches to graphics mode and sets the pointers for the graphics primitives.

See also
Herc720x348x2, EGA640x350x16, VGA640x480x16.

EGA640x350x16 Procedure — FASTGRPH

Function
Sets the video mode to 640 x 350 in 16 colors.

Declaration
EGA640x350x16

Remarks

See also
VGA640x480x16, Graph Unit

Herc720x200x2 Procedure — FASTGRPH

Function
Sets the video mode to 720 x 200 in 2 colors.

Declaration
Herc720x348x2

Remarks
This procedure switches to graphics mode and sets the pointers for the graphics primitives.

See also
CGA640x200x2, EGA640x350x16, VGA640x480x16.

SetVideoChoices FASTGRPH

Function

Sets the allowable video modes.

Declaration

SetVideoChoices(VMode : Word; Accept : Boolean);

Remarks

By default all video modes are acceptable. Certain programs may not support all video modes.

See also

VideoID, VideoAutoDetect.

Example

This statement would cause the program to abort if it were run on a machine which only supported CGA graphics.

```
  SetVideoChoices(TG_CGA,FALSE);
```

SVGA800x600x16 FASTGRPH

Function

Sets the video mode to 800 x 600 in 2 colors.

Declaration

SVGA800x600x16;

Remarks

Requires hardware and screen that provide super VGA resolutions.

See also

CGA640x200x2, EGA640x350x16, Herc720x348x2,
  VGA640x480x16

VGA640x480x16 Procedure FASTGRPH

Function

Sets the video mode to 640 x 480 in 16 colors.

Declaration

VGA640x480x16

Remarks

This procedure switches to graphics mode and sets the pointers for the graphics primitives.

Restrictions

Requires a VGA card and monitor.

See also

CGA640x200x2, EGA640x350x16, Herc720x348x2.

VideoAutoDetect FASTGRPH

Function

Detects the graphics equipment and switches to graphics mode if available.

Declaration

VideoAutoDetect;

Remarks

Selects the highest resolution that is available and supported.

The global variable InitDriverCode can be examinded to determine the video mode set.

See also

VideoID

VideoID FASTGRPH

Function

Detects the graphics equipment available.

Declaration

VideoID(VAR v : VidID);

Remarks

Graphics equipment is only detected. The current video mode is not changed.

Graphic Primitives

Turbo Pascal offers a rich set of graphics commands, that work with almost any video display. However, the drawback to the flexibility of Turbo Pascal's BGI Graphics is the speed at which the graphics are displayed.

To provide a toolset that could operate quickly, the following assembler graphic routines were written to replace the ones offered by TP.

Other then the documented restrictions you may freely mix and match Turbo's graphic routines with TEGL's.

The following constants are defined in the FASTGRPH unit and may be assigned to RMWBITS to define the type of binary operation between each byte in the line and the corresponding bytes on the screen.

```
VAR
    RMWBITS    : WORD;

TYPE
    FGNORM     = 0;
    FGAND      = $08;
    FGOR       = $10;
    FGXOR      = $18;
    FGNOT      = $80;
```

FastLine Procedure — FASTGRPH

Function

Draws a line from (x,y) to (x1,y2).

Declaration

FastLine (x,y,x1,y2,n:word)

Remarks

Sets the global variable RMWBITS to the appropriate mode for drawing the line.

x,y specifies the line starting coordinates.

x1,y1 specifies the line ending coordinates.

n specifies the color of the line.

Fastline will only draw a continuous line. SetLineStyle, SetColor and SetWriteMode has no effect on Fastline.

See also

Turbo Pascal Reference Manual sh Line

Putpixs Procedure FASTGRPH

Function

Plots a pixel at x,y.

Declaration

PutPixs (x,y,n:word)

Remarks

Plots a point in the color defined by n at (x,y).

Set the global variable RMWBITS to the appropriate mode for plotting the pixel.

Putpixs replaces the PutPixel routine in the Graph Unit.

See also

Getpixs, Turbo Pascal Reference Manual sh PutPixel

Getpixs Function FASTGRPH

Function

Return the pixel value at x,y.

Declaration

Getpixs (x,y:word)

Result type

Word.

Remarks

Gets the pixel color at (x,y).

Getpixs replaces the GetPixel routine in the Graph Unit.

See also

Putpixs, Turbo Pascal Reference Manual sh GetPixel

Getbiti Procedure FASTGRPH

Function

Copies the specified screen image into a buffer.

Declaration

Getbiti(x,y,x1,y1:word;buffer:pointer)

Remarks

x,y,x1,y1 defines a rectangular region on the screen.

buffer is a memory area that may be allocated by GetMem or TEGLGetMem.

Getbiti replaces the GetImage routine in the Graph Unit. By using TEGLGetmem with BigImageSize, Getbiti will allow saving of images larger than 64k.

Restrictons

The saved image structure of Getbiti and Putbiti is different than what Turbo Pascal's GetImage and PutImage use.

See also

Putbiti, BigImageSize

Putbiti Procedure FASTGRPH

Function

Copies the buffer to the specified screen area.

Declaration

Putbiti(x,y : word; buffer : pointer; RMWbits : word)

Remarks

x,y defines the upper left corner of the screen area for placing the saved image.

buffer is the image buffer that contains a copy of the screen image saved previously by Getbiti.

RMWbits defines the type of binary operation between the saved image and the corresponding bytes on the screen.

Putbiti replaces the PutImage routine in the Graph Unit. By using TEGLGetmem with BigImageSize, Putbiti will allow the saving and restoring of images larger then 64k.

Restrictons

The saved image structure of Getbiti and Putbiti is different than what Turbo Pascal's GetImage and PutImage use.

See also

Getbiti, BigImageSize

BigImageSize Function FASTGRPH

Function

Calculates the size of the image buffer.

Declaration

BigImageSize(x,y,x1,y1:word) : LongInt

Result type

Longint.

Remarks

x,y,x1,y1 defines the rectangular coordinates that will be used for Getbiti.

BigImageSize replaces Turbo Pascal's ImageSize routine. By using TEGLGetmem with BigImageSize, image buffers may be larger then 64k.

See also

Getbiti, Putbiti

SetAPage Procedure FASTGRPH

Function

Sets the active page for graphics output.

Declaration

SetAPage(pagenum:word)

Remarks

Makes pagenum the active graphics page. All output, including those from Turbo Pascal's graphics routines, will be directed to pagenum.

Only two pages are supported with the EGA's 640 x 350 x 16 mode.

SetAPage replaces the Turbo Pascal's SetActivePage procedure.

See also

SetVPage, FlipAPage, FlipVPage, VideoPage

SetVPage Procedure — FASTGRPH

Function

Sets the visual graphics page number.

Declaration

SetVPage(pagenum:word)

Remarks

Makes pagenum the visual graphics page. All output, including that from Turbo Pascals's graphics routines, will still be directed to the active pagenum.

Only two pages are supported with the EGA's 640 x 350 x 16 mode.

SetVPage replaces the TP's SetVisualPage procedure.

See also

SetAPage, FlipAPage, FlipVPage, VideoPage

FlipAPage Procedure — FASTGRPH

Function

Flips the active page to the alternate page.

Declaration

FlipAPage

Remarks

Makes the alternate page the active graphics page. All output, including that from Turbo Pascal's graphics routines, will be directed to the new active page.

Only two pages are supported with the EGA's 640 x 350 x

16 mode. If the current active page is (1), FlipAPage will set the active page to (2). The reverse is true, if the current active page is (2).

FlipAPage does not have an equivalent in the Graph Unit.

See also

SetAPage, SetVPage, FlipVPage, VideoPage

FlipVPage Procedure — FASTGRPH

Function

Flips the visual page to the alternate page.

Declaration

FlipVPage

Remarks

Makes the alternate page the visual graphics page.

Only two pages are supported with EGA's 640 x 350 x 16. If the current visual page is (1), FlipVPage will set the visual page to (2). The reverse is true, if the current visual page is (2).

FlipVPage does not have an equivalent in the Graph Unit.

See also

SetAPage, SetVPage, FlipAPage, VideoPage

VideoPage Function — FASTGRPH

Function

Returns the current Visual page.

Declaration

VideoPage

Result type

Word.

Remarks

Returns the current visual graphics page.

Only two pages are supported with the EGA's

640 x 350 x 16 mode.

VideoPage does not have an equivalent in the Graph Unit.

See also

SetAPage, SetVPage, FlipAPage, FlipVPage

New Graphic Primitives

The TEGL Windows Tookit's ability to display fast graphics is, in a way, just the tip of the iceberg. The following routines provide functions to extract and overlay buffered images before displaying the final results on the screen.

Some of these routines may be used to create a virtual image (an image larger then the size of the screen). The only limitation at this time is the need for graphic primitives that will draw to a buffered image.

Extractpixs Function — FASTGRPH

Function

Return the pixel value at x,y within an image buffer.

Declaration

Extractpixs (x,y:word; buffer:pointer)

Result type

Word

Remarks

Gets the pixel color at (x,y) within the saved image buffer.

ExtractIMG Procedure — FASTGRPH

Function

Extracts an image area x,y,x1,y1 from buff2 to buff1.

Declaration

ExtractIMG(x,y,x1,y1:word;buff1,buff2:pointer)

Remarks

Returns a partial image in buff1 from buff2.

See also

OverlayIMG, PutBiti, GetBiti

OverlayIMG Procedure FASTGRPH

Function

Overlays image buff1 to buff2 at x,y offsets.

Declaration

OverlayIMG(x,y:word;buff1,buff2:pointer)

Remarks

Overlays an image in buff1 to buff2.

See also

ExtractIMG, PutBiti, GetBiti

SwapBytes Procedure FASTGRPH

Function

Swaps two buffers.

Declaration

SwapBytes(buff1,buff2:pointer; bytestoswap:longint)

Remarks

Swaps the images within buff1 with buff2.

Graphic Derivatives

The following are some fast common routines to create XOR boxes that can be erased simply by calling the routine again.

XORing pixels to the screen has the unique feature that when the same pixel is XORed to the same location a second time the pixel is restored to it's original look.

The XOR box routines here allow boxes to flit and dance across the screen

without (if used correctly) changing any of the underlying display.

XORCornerBox Procedure FASTGRPH

Function
Creates box corners only.

Declaration
XORCornerBox (x,y,x1,y1,color : integer)

Remarks
x,y,x1,y1 are the coordinates of a rectangle.

This routine is used in Ziptobox and Zipfrombox to create the shrinking and expanding corner images.

XORBox Procedure FASTGRPH

Function
Draws a (xor) rectangle.

Declaration
XORBox (x,y,x1,y1,color : integer)

Remarks
(x,y) define the upper left corner of a rectangle, and (x1,y1) define the lower right corner. Coordinates must be within the physical screen.

This routine is used in MoveFrame to move an (xor) box image around.

Icon Graphics

Putpict Procedure FASTGRPH

Function
Puts an icon to a specified screen area.

Declaration

Putpict (x,y:word; buf:pointer;n:word)

Remarks

x,y defines the upper left corner of the screen area for placing the icon image.

buf points to the icon image.

n is the default color for any pixel that is black within the icon.

Restrictions

Icons are stored in a unique fashion, these are not bit images in the conventional sense. Icons are created and maintained using the Icon Editor and support programs.

See also

PictSize, Icon Editor.

PictSize Procedure FASTGRPH

Function

Gets the width and height in pixels of an icon image.

Declaration

PictSize(VAR Width,Height: Word; Buffer: Pointer);

Remarks

Buffer must point to a valid icon image.

See also

PutPict, Icon Editor.

Abort Procedure FASTGRPH

Function

Closes the graphics system and displays the message string.

Declaration

Abort(Msg : string)

Remarks

This routine is defined in Fastgrph because of the need for closing the graphics system and returning to text mode before the message can be displayed.

Chapter 7 - Assembly Language Graphics

Chapter 8 - Special Effects

Special Effects

The TEGLGrph unit has a nice collection of graphic effects that may be used to create 3D characters, shadow boxes, long icon buttons, etc..

These routines may be combined with Turbo Pascal's BGI fonts and graphics for ev more effects.

We suggest that if you build other graphic effects they should support a standard parameter list. Specifically coordinates should be ordered x, y, x1, y where x, y are the upper left coordinates and x1, y1 are the lower right coordinates of an area on the screen.

Screen Backdrop

The backdrop is normally the full physical screen filled with a color and pattern to give the effect of a mat. On this mat we place icons and open up windows. It's like the velvet mat a Jeweler uses to show off gem stones.

The backdrop does not require a window frame to draw on.

ClearTEGLScreen Procedure — TEGLGRPH

Function

Clears the screen to the backdrop pattern.

Declaration

ClearTEGLScreen

Remarks

Fills the complete screen using the bitmask found in TEGLBackPattern or TEGLFillStyle with the background color of TEGLBackColor. Completes the clearing by placing a border if TEGLBorderShow is TRUE in the color of TEGLBorderColor.

The default is a gray matted area with white borders.

Restrictions

Must be in Graphics mode.

See also

SetTEGLBorderShow, SetTEGLBackColor, SetTEGLBorderColor, SetTEGLFillPattern, SetTEGLFillStyle

Example

```
  EGA640x350x16;        { -- Sets the graphics mode }
  SetMouseMinMax(0,0,GetMaxX,GetMaxY);

  ClearTEGLScreen;
```

SetTEGLBorderShow Procedure TEGLGRPH

Function

Sets the switch on whether a border should be drawn or not drawn after the bar fill.

Declaration

SetTEGLBorderShow(BorderShow:boolean)

Remarks

Switches the border on=TRUE or off=FALSE when TEGLClearScreen is called.

The default is on TRUE.

Restrictions

Must be called before calling TEGLClearScreen.

See also

TEGLClearScreen, SetTEGLBackColor, SetTEGLBorderColor, SetTEGLFillPattern, SetTEGLFillStyle

Example

```
  SetTEGLBorderShow(FALSE);
  ClearTEGLScreen;
```

SetTEGLBackColor Procedure TEGLGRPH

Function

Sets the color of the backdrop.

Declaration

SetTEGLBackColor(BackColor:word)

Remarks

Sets the background color for the backdrop to BackColor.

The default is WHITE.

Restrictions

Must be called before calling TEGLClearScreen.

See also

TEGLClearScreen, SetTEGLBorderShow, SetTEGLBorderColor, SetTEGLFillPattern, SetTEGLFillStyle

Example

```
  SetTEGLBackColor(GREEN);
  ClearTEGLScreen;
```

SetTEGLBorderColor Procedure TEGLGRPH

Function

Sets the border color of the backdrop.

Declaration

SetTEGLBorderColor(BorderColor:word)

Remarks

Sets the border color for the backdrop to BorderColor.

The default is WHITE.

Restrictions

Must be called before calling TEGLClearScreen.

See also

TEGLClearScreen, SetTEGLBorderShow, SetTEGLBackColor, SetTEGLFillPattern, SetTEGLFillStyle

Example

```
SetTEGLBorderColor(BROWN);
ClearTEGLScreen;
```

SetTEGLFillPattern Procedure TEGLGRPH

Function

Sets the Fill pattern for the backdrop.

Declaration

SetTEGLFillPattern(backpattern:FillPatternType)

Remarks

Sets the fill pattern for the backdrop to backpattern.

The default is defined as a constant:

```
CONST
  TEGLBackPattern : FillPatternType =
  ($AA,$55,$AA,$55,$AA,$55,$AA,$55);
```

Restrictions

Must be called befor calling TEGLClearScreen.

See also

TEGLClearScreen, SetTEGLBorderShow, SetTEGLBackColor, SetTEGLBorderColor, SetTEGLFillStyle

Example

```
CONST
   MyPattern : FillPatternType =
     ($FF,$22,$FF,$22,$FF,$22,$FF,$22);

  SetTEGLFillPattern(MyPattern);
  ClearTEGLScreen;
```

SetTEGLFillStyle Procedure TEGLGRPH

Function

Sets the Fill style for the backdrop.

Declaration

SetTEGLFillStyle(pattern:word)

Remarks

Sets the fill style to pattern.

Use one of the predefined fill styles in the Graph Unit.

Setting the fill style cancels the user defined pattern.

Restrictions

Must be called before calling TEGLClearScreen.

See also

TEGLClearScreen, SetTEGLBorderShow, SetTEGLBackColor, SetTEGLBorderColor, SetTEGLFillPattern

Example

```
  SetTEGLFillPattern(SolidFill);
  ClearTEGLScreen;
```

Creating Shadow Boxes

A shadow box is a simple rectangular that has a shadow edge to give a 3-dimensional effect. A shadow box is the quickest method to clear a window after PushImage.

ShadowBox Procedure — TEGLGRPH

Function

Creates a 3-D type box at the rectangular area defined by x, y, x1, y1.

Declaration

ShadowBox(x,y,x1,y1 : word)

Remarks

x, y, x1, y1 defines the rectangular area for the ShadowBox.

The default Bar SOLID fill color is WHITE with BLACK borders and BLACK shadow.

See also

SetShadowColor, SetShadowBorderColor,

SetShadowFillPattern, SetShadowFillStyle

Example

```
PushImage(100,100,200,200);
ShadowBox(100,100,200,200);
```

ShadowBoxText Procedure TEGLGRPH

Function

Outputs a text string within a ShadowBox.

Declaration

ShadowBoxText(x,y,txtlen:word; textstr:string)

See also

ShadowBox

Example

```
ShadowBoxText(100,100,200,'TEGL Systems Corporation');
```

SetShadowColor Procedure TEGLGRPH

Function

Sets the Bar fill color.

Declaration

SetShadowColor(bcolor:word)

Remarks

bcolor defines the ShadowBox color.

The default Bar fill color is WHITE.

See also

ShadowBox, SetShadowBorderColor,
SetShadowFillPattern, SetShadowFillStyle

Example

```
PushImage(100,100,200,200);
  SetShadowColor(red);
  ShadowBox(100,100,200,200);
```

SetShadowBorderColor Procedure TEGLGRPH

Function

Sets the ShadowBox border color.

Declaration

SetShadowBorderColor(bcolor:word)

Remarks

bcolor defines the ShadowBox border color.

The default border color is BLACK.

See also

ShadowBox, SetShadowColor, SetShadowFillPattern, SetShadowFillStyle

Example

```
  PushImage(100,100,200,200);
  SetShadowBorderColor(LIGHTGRAY);
  ShadowBox(100,100,200,200);
```

SetShadowFillPattern Procedure TEGLGRPH

Function

Sets the bar fill pattern for ShadowBox.

Declaration

SetShadowFillPattern(backpattern: FillPatternType)

Remarks

backpattern is of the type FillPatternType

```
CONST
  MyShadowPattern : FillPatternType =
    ($AA,$55,$AA,$55,$AA,$55,$AA,$55);
```

The default fill pattern is SOLIDFILL which is defined in the Graph Unit.

See also

ShadowBox, SetShadowColor, SetShadowBorderColor, SetShadowFillStyle

Example

```
CONST
  MyShadowPattern : FillPatternType =
    ($AA,$55,$AA,$55,$AA,$55,$AA,$55);

  PushImage(100,100,200,200);
  SetShadowFillPattern(MyShadowPattern);
  ShadowBox(100,100,200,200);
```

SetShadowFillStyle Procedure — TEGLGRPH

Function

Sets the bar fill style for ShadowBox.

Declaration

SetShadowFillStyle(pattern:word)

Remarks

pattern is of one of the predefined type in TP's Graph unit.

The default fill style is SOLIDFILL.

See also

ShadowBox, SetShadowColor, SetShadowBorderColor, SetShadowFillPattern

Example

```
PushImage(100,100,200,200);
SetShadowFillStyle(LineFill);
ShadowBox(100,100,200,200);
```

Creating Shadow Text

Shadow text enhances the normal BGI fonts by writing the text string several times with a slight shift of the x,y coordinates on each write.

This simple method provides a 3-D quality to any BGI or TEGL font.

Shadowtext Procedure — TEGLGRPH

Function

Displays a shadowed textstr at (x,y).

Declaration

Shadowtext(x,y,color:word; textstr:string)

Remarks

x,y specifies the coordinates for displaying the textstr.

color specifies the color of the textstr.

Shadowtext is affected by SetTextStyle, SetTextJustify and SetUserCharSize in the Graph Unit.

See also

SetShadowTextType, SetShadowTextShadow, SetShadowTextHighlight, ShadowTextHighlightOFF

Example

```
  ShadowText(100,100,LightCyan,'TEGL Systems Corporation');
```

SetShadowTextType Procedure — TEGLGRPH

Function

Sets the shadow text font type.

Declaration

SetShadowTextType(texttype:pointer)

Remarks

texttype is a pointer to one of the TEGL fonts. If texttype is set to nil, ShadowText uses OutTextXY in the Graph Unit.

See also

ShadowText, SetShadowTextShadow, SetShadowTextHighlight, ShadowTextHighlightOFF

Example

```
SetShadowTextType(@Script);
ShadowText(100,100,LightCyan,'TEGL Systems Corporation');
```

SetShadowTextShadow Procedure TEGLGRPH

Function

Sets the shadow color for ShadowText.

Declaration

SetShadowTextShadow(color:word)

Remarks

color is the shadow color when displaying the shadowed text.

The default shadow color is BLACK.

See also

ShadowText, SetShadowTextType, SetShadowTextHighlight, ShadowTextHighlightOFF

Example

```
SetShadowTextShadow(lightgray);
ShadowText(100,100,LightCyan,'TEGL Systems Corporation');
```

SetShadowTextHighlight Procedure TEGLGRPH

Function

Sets the highlighted color for ShadowText.

Declaration

SetShadowTextHighlight(color:word)

Remarks

color is the highlighted color when displaying the shadowed text. Normally, ShadowText toggles the high bit of color to achieve the different shadings.

See also

ShadowText, SetShadowTextType, SetShadowTextShadow, ShadowTextHighlightOFF

Example

```
  SetShadowTextHighlight(blue);
  ShadowText(100,100,LightCyan,'TEGL Systems Corporation');
```

ShadowTextHighlightOFF Procedure TEGLGRPH

Function

Resets the highlight color set by SetShadowTextHighlight.

Declaration

ShadowTextHighlightOFF

Remarks

Switches off the highlight color set by SetShadowTextHighlight.

See also

ShadowText, SetShadowTextType, SetShadowTextShadow, SetShadowTextHighlight

Example

```
  SetShadowTextHighlight(blue);
  ShadowText(100,100,LightCyan,'TEGL Systems Corporation');
  ShadowTextHighlightOFF;
  ShadowText(100,120,LightCyan,'TEGL Systems Corporation');
```

Other text effects

ExtendTextXY Procedure TEGLGRPH

Function

Makes embossed text.

Declaration

ExtendTextXY(X,Y : Word; S : String);

Restrictions

Does not work with BGI fonts.

Example

```
VAR ifs : ImageStkPtr;

  QuickFrame(ifs,100,100,300,150);
  OutTEGLTextXY(105,105,'Normal Text');
  ExtendTextXY(105,125,'FAT TEXT');
```

ShiftTextXY Procedure TEGLGRPH

Function

Writes text with a leading white edge.

Declaration

ShiftTextXY(X,Y : Word; S : String);

Restrictions

Does not work with BGI fonts.

Remarks

X and Y are absolute screen coordinates, S is the string to display.

Example

```
VAR ifs : ImageStkPtr;

  SetShadowColor(LightGray);
  QuickFrame(ifs,100,100,300,150);
  OutTEGLTextXY(105,105,'Normal Text');
```

```
ShiftTextXY(105,125,'Shifted Text');
```

Buttons

DefineButtonClick Procedure — TEGLGRPH

Function

Displays an icon, sets mouse click area and attaches it to an Event.

Declaration

```
DefineButtonClick(ifs : ImageStkPtr; x,y : Word;
  button : Pointer; p : CallProc);
```

Remarks

Ifs is the frame the icon is placed on. Button can be any icon image. P is the Event to pass control to when the icon is clicked on.

P can be set to CollapseToIconShow or CollapseToMsClick if the button is for closing a frame.

Example

```
DefineButtonClick(ifs,150,200,@ImageOK,CollapseToIconShow);
```

DefineLongButtonClick Procedure — TEGLGRPH

Function

Displays a long button with text, sets mouse click area, and attaches it to an event.

Declaration

```
DefineLongButtonClick(ifs : ImageStkPtr; x,y,ln : Word;
  msg : String; p : CallProc);
```

Remarks

Ifs is the frame the button is placed on. x,y are the

coordinates to place the button at. Ln is the length of the message in pixels (depends on currently selected font) and msg is the text to place inside the button. P is the event to activate when the button is clicked on.

Example

```
DefineLongButtonClick(ifs,100,150,35,'Quit',CollapseToMsClick);
```

DefineUserButtonClick Procedure TEGLGRPH

Function

Displays a button with text, sets mouse click area, and attaches it to an event.

Declaration

```
DefineUserButtonClick(ifs : ImageStkPtr; x,y : Word;
  msg : String; p : CallProc);
```

Remarks

Ifs is the frame the button is placed on. x,y are the coordinates to place the button at and msg is the text to place inside the button. P is the event to activate when the button is clicked on.

Restrictons

Msg cannot be more than about 4 characters. This is dependant on the currently selected font.

Example

```
DefineUserButtonClick(ifs,100,150,'Quit',CollapseToMsClick);
```

PutUserButtonClick Procedure TEGLGRPH

Function

Draws a button at the coordinates with a message.

Declaration

PutUserButtonClick(ifs : ImageStkPtr; x,y : Word;
  msg : String)

Restrictions

Msg cannot be more than about 4 charcters, depends upon the currently selected font.

Remarks

This routine just displays a button, no mouse click area is defined.

Explosions

CollapseToIconShow Event TEGLGRPH

Function

Collapse a frame and restore the icon it came from.

Declaration

CollapseToIconShow(ifs : ImageStkPtr; ms : MsClickPtr) :
  Word;

Restrictions

Should only be attached to a frame created after a call to ExplodeToIconShow.

Remarks

After opening a frame from a ExplodeToIconShow, this Event can be attached to a button within the frame. When this button is clicked on, the frame will collapse and zip to the original icon location and restore the icon.

See also

ExplodeToIconShow,DefineButtonClick.

CollapseToMsClick Event TEGLGRPH

Function

Collapse a frame and zip back to the original mouse click position.

Declaration

CollapseToMsClick(ifs : ImageStkPtr; ms : MsClickPtr):
   Word;

Restrictions

Should only be attacted to a frame created after a call to ExplodeFromMsClick.

Remarks

After opening a frame from a ExplodeFromMsClick, this Event can be attached to a button within the frame. When this button is clicked on, the frame will collapse and zip to the original defined mouse click area.

See also

ExplodeFromMsClick, DefineButtonClick.

ExplodeFromIconHide TEGLGRPH

Function

Hides the icon, zips and opens a new frame.

Declaration

```
ExplodeFromIconHide(ifs : ImageStkPtr; ms: MouseClickPtr;
  x,y,x1,y1 : Word);
```

Restrictions

The icon exploded from must be in a frame of its own for this to look right.

Remarks

ifs and ms are the parameters passed to an event. x,y,x1,y1 are the coordinates where a new frame is to be opened. After a call to this procedure a new frame is created. Save the Global Variable StackPtr if you wish to manipulate the new frame.

See also

CollapseToIconShow, DefineButtonClick.

ExplodeFromMsClick Procedure TEGLGRPH

Function

Zips from a mouse click location to a new frame position.

Declaration

```
ExplodeFromMsClick(ifs : ImageStkPtr; ms : MouseClickPos;
  x,y,x1,y1 : Word);
```

Remarks

ifs and ms are the parameters passed to an event. x,y,x1,y1 are the coordinates where a new frame is

to be opened. After a call to this procedure a new frame is created. Save the Global Variable StackPtr if you wish to manipulate the new frame.

See also

CollapseToMsClick, DefineUserButtonClick.

Moving and Transforming XOR Boxes

MoveBox Procedure TEGLGRPH

Function

Moves a (XOR) wire frame from x, y to ax, ay.

Declaration

MoveBox(ax,ay,x,y,x1,y1 : integer)

Remarks

x, y, x1, y1 specify the coordinates of the starting (XOR) wire frame.

ax, ay are the upper left coordinates of the ending position of the (XOR) wire frame.

The box movement is divided into 6 steps which is added or subtracted from the originating position until it reaches the destination.

The global variable ZipDuration may be changed to set the delay between each movement step.

See also

XORBox, XORCornerBox, ZipToBox, ZipFromBox

Example

A wire frame box 50(w) x 50(h) is moveed from 100,100 to 500,280.

```
  MoveBox(500,280,100,100,150,150);
```

ZipToBox Procedure TEGLGRPH

Function

Creates a moving and expanding (XOR) wire frame from ax, ay, ax1, ay1 to x, y, x1, y1.

Declaration

ZipToBox(ax,ay,ax1,ay1,x,y,x1,y1 : integer)

Remarks

ax, ay, ax1, ay1 specifies the rectangular coordinates of the starting (XOR) wire frame.

x, y, x1, y1 specifies the rectangular coordinates of the ending (XOR) wire frame.

The box is moved from (ax,ay) to (x,y) using MoveBox before the box is transformed (expanded). The transformation is divided into 6 steps which is added or subtracted from (ax,ay,ax1,ay1) until the size equals (x,y,x1,y1).

The global variable ZipDuration may be changed to set the delay between each movement step.

See also

XORBox, XORCornerBox, MoveBox, ZipFromBox

Example

A wire frame box 50(w) x 50(h) at (100,100) will be visually moved and expanded to a box 100(w) x 100(h) at 400,200.

```
ZipToBox(100,100,150,150,400,100,500,200);
```

ZipFromBox Procedure TEGLGRPH

Function

Creates a shrinking and moving (XOR) wire frame from x, y, x1, y1 to ax, ay, ax1, ay1.

Declaration

ZipFromBox(ax,ay,ax1,ay1,x,y,x1,y1 : integer)

Remarks

x, y, x1, y1 specifies the rectangular coordinates of the starting (XOR) wire frame.

ax, ay, ax1, ay1 specifies the rectangular

coordinates of the ending (XOR) wire frame.

The box is transformed by dividing the transformation steps into 6 steps which is added or subtracted from (x, y,x1,y1) until the size equals (ax,ay,ax1,ay1). The box is then moved from (x,y) to (ax,ay) using MoveBox.

The global variable ZipDuration may be changed to set the delay between each movement step.

See also

XORBox, XORCornerBox, MoveBox, ZipFromBox

Example

A wire frame box 100(w) x 100(h) at (x=400,y=200) will be visually shrunk and moved to a box 50(w) x 50(h) at (x=100,y=100).

```
ZipFromBox(100,100,150,150,400,100,500,200);
```

Icon Button

DrawLongButton Procedure TEGLGRPH

Function

Creates an icon button of size ln at (x,y).

Declaration

DrawLongButton(x,y,ln : word)

Remarks

x,y specifies the coordinates for the icon button. ln specifies the length of the icon button in pixels.

Example

```
DrawLongButton(x,y,200);
fonttable := @font14;
setcolor(white);
outtegltextxy(x+15,y+1,'TEGL Systems Corporation');
```

Writing Events

All Event-handlers must use the following header definition.

```
{$F+}
function MyEvents(Frame:imagestkptr; MouseClickPos: msclickptr) : word;
{$F-}
```

This is the declaration of a CallProc. Note the far call directive. If you write an event that does not used the far call directive you will be unable to use it as a parameter. The compiler will give an error message 143 of q Invalid procedure or function reference.

Mouse Awareness

FindFrame Function — TEGLUNIT

Function

Searches through the Frame stack for the first frame that overlaps the coordinates passed as a parameter.

Declaration

FindFrame(mxpos,mypos:word)

Result type

Pointer.

Remarks

Returns a ImageStkPtr if the parameters overlap one of the frames, otherwise returns Nil for no match.

FindFrame is used by the TEGLSupervisor, but is provided as an external procedure to allow for specialize routines that may be used to replace the TEGLSupervisor.

Restrictions

FindFrame starts the scan from the top of the stack, thereby returning the first frame found that overlaps the parameters.

See also

CheckMouseClickPos

Example

The following example creates 250 random boxes and monitors the position of the mouse pointer to see if it

overlaps one of the boxes. The timer tick routine is used to blink the overlapped box, once every second.

```
Var i       : word;
    fs1,fs2  : imagestkptr;

function BlinkBox(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word;
   BEGIN
      if fs1<>nil then
        begin
           hidemouse;

           If fs1^.ImageActive then
              hideimage(fs1)
           else
              Showimage(fs1,fs1^.x,fs1^.y);

           showmouse;
        end;

      beep(10000,1,1);
      ResetTimerFlag;
      BlinkBox := 1;
   end;

procedure CreateRandomBox(x,y:word);
   begin
      PushImage(x,y,x+20,y+20);
      shadowbox(x,y,x+20,y+20);
   end;

for i:=1 to 250 do
   createrandombox(random(600),random(320));

fs1 := stackptr;
SetTimerTick(18,BlinkBox);
repeat
   if mouse_buttons<>0 then
      fs2 := frameselectandmove(mouse_xcoord,mouse_ycoord)
   else
      fs2 := findframe(mouse_xcoord,mouse_ycoord);

   if (fs2<>nil) and (fs1<>fs2) then
      begin
        If (not fs1^.ImageActive) then
           i:=blinkbox(nil,nil);
        fs1 := fs2;
      end;

   checkctrlbreak;
```

```
until false;
```

CheckMouseClickPos Function TEGLUNIT

Function

Compares all Mouse click defines within a frame, for a match with the current mouse coordinates.

Declaration

CheckMouseClickPos(Frame,mxpos,mypos:word)

Result type

Pointer.

Remarks

Returns a MSClickPtr type if mouse coordinates matches one of the mouse click defines, otherwise returns Nil for no match.

CheckMouseClickPos is normally an internal procedure, used by the TEGLSupervisor. The Mouse Click position information is normally provided as the second parameter of an event, whenever an event is called.

However, CheckMouseClickPos may be used to rewrite the TEGLSupervisor or used to determine if the Mouse Click position has changed.

Restrictions

FindFrame should be used first, to check if another frame is overlapping the current frame, before using CheckMouseClickPos.

See also

DefineMouseClickPtr, ResetMouseClicks, FindMouseClickPtr, ResetMSClickSense, ResetMSClickCallProc, ResetMSClickActive

Example

The following example defines an array of 100 Mouse Click Areas which uses CheckMouseClickPos to establish the mouse location within the frame.

```
var x,y : word;

Function PlayAllNotes(Frame:ImageStkPtr;
          MouseClickPos: MSClickPtr):WORD;
   var ms   : msclickptr;
   BEGIN
```

```
      while findframe(mouse_xcoord,mouse_ycoord)=frame do
        begin
           ms := CheckMouseClickPos(frame,mouse_xcoord,
                   mouse_ycoord);
           if ms<>nil then
              sound(ms^.clicknumber*10);
        end;

      nosound;
      PlayAllNotes := 0;
   END;

PushImage(1,1,107,124);
shadowbox(1,1,107,124);
DefineMouseClickArea(stackptr,1,1,107,124,true,PlayAllNotes,MSSense);

for x:=0 to 9 do
   for y:=0 to 9 do
      begin
        shadowbox(stackptr^.x+3+x*10,stackptr^.y+3+y*10,
          stackptr^.x+3+10+x*10,stackptr^.y+3+10+y*10);
        DefineMouseClickArea(stackptr,3+x*10,3+y*10,3+x*10+6,
          3+y*10+6,true,nilunitproc,MSClick);
      end;
```

CheckForMouseSelect Function — TEGLUNIT

Function

Checks if one of the mouse click areas within a frame has been selected.

Declaration

CheckforMouseSelect(frame)

Result type

Returns the Mouse Click Pointer if mouse button was released while the mouse cursor overlaps a button icon.

Remarks

This procedure may be used when only the Frame is known and the program is waiting for the user to click on one of a series of unknown icons.

CheckForMouseSelect may be used within an event to wait

on a multiple button type icon replies from the user.

If PressButtonFlag is true, then visualbuttonpress is called to simulate the pressing of a button icon.

Restrictions

If PressButtonFlag is true, the restrictions for VisualButtonPress should be followed. If the icon does not have a black fringe, set PressButtonFlag to false.

See also

PressButton, VisualButtonPress

Example

The following example creates (8) button type icons, which calls up a window that displays two choices, Cancel or OK. The event waits until one of the choices are made before returning to TEGLSupervisor.

```
var x,y : word;

function DemoCancelOK(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word;
   begin
      if visualbuttonpress(frame,MouseClickPos) then
        begin
           hidemouse;

           PushImage(frame^.x,frame^.y,frame^.x+100,frame^.y+50);
           shadowbox(frame^.x,frame^.y,frame^.x+100,frame^.y+50);

           Putpict(frame^.x+6,frame^.y+6,@imageCancel,black);
           DefineMouseClickArea(stackptr,6,6,6+35,6+12,true,
             nilunitproc,MSClick);

           Putpict(frame^.x+12+35,frame^.y+6,@imageOK,black);
           DefineMouseClickArea(stackptr,12+35,6,12+35+35,6+12,
             true,nilunitproc,MSClick);

           showmouse;
           repeat
              mouseClickPos := CheckforMouseSelect(stackptr);
           until MouseClickPos<>nil;

           if MouseClickPos^.Clicknumber=1 then
              SlideBeep(100,500,3)
           else
              Beep(800,3,100);

           hidemouse;
           popimage;
           showmouse;
        end;
```

```
      DemoCancelOK := 1;
   end;

PushImage(1,1,100,100);
shadowbox(1,1,100,100);

for x:=0 to 1 do
   for y:=0 to 4 do
      begin
        Putpict(stackptr^.x+6+x*42,stackptr^.y+6+y*18,
          @imageBlankBut,black);
        DefineMouseClickArea(stackptr,5+x*42,5+y*18,5+x*42+35,
          5+y*18+12,true,DemoCancelOK,MSClick);
      end;
```

Special Effects

PressButton Procedure TEGLUNIT

Function

Simulates the pressing of a button type icon. The actual routine simply shifts the icon down and to the right by two pixels.

Declaration

PressButton(fs,mouseopt:msclickptr)

Remarks

This procedure is used mainly by VisualButtonPress to simulate the action of a electronic button switch.

PressButton may be used to create the illusion of a button left in the down position.

Restrictions

You are required to redraw the button if you need the button in the up position.

This routine only works with icons that have a black fringe of two pixels wide on the right and bottom of the icon. The defined mouse click area should not include this shadow area ie. x1 and y1 is less two pixels.

See also

VisualButtonPress, CheckForMouseSelect

Example

The following example creates (8) button type icons and toggles the buttons on/off whenever the icon is clicked upon.

```
var x,y : word;

function SwitchOff(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word; forward;

function SwitchOn(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word;
   begin
      Beep(1500,1,10);
      PressButton(Frame,MouseClickPos);
      ResetMSClickCallProc(Frame,MouseClickPos^.ClickNumber,SwitchOff);
      while Mouse_Buttons<>0 do;
      SwitchOn := 1;
   end;

function SwitchOff(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word;
   begin
      Beep(1500,1,10);
      hidemouse;
      Putpict(Frame^.x+MouseClickPos^.ms.x,
             Frame^.y+MouseClickPos^.ms.y,@imageBlankBut,black);

      showmouse;
      ResetMSClickCallProc(Frame,MouseClickPos^.ClickNumber,
        SwitchOn);
      while Mouse_Buttons<>0 do;
      SwitchOff := 1;
   end;

PushImage(1,1,100,100);
shadowbox(1,1,100,100);

for x:=0 to 1 do
   for y:=0 to 4 do
      begin
        Putpict(stackptr^.x+6+x*42,stackptr^.y+6+y*18,
          @imageBlankBut,black);
        DefineMouseClickArea(stackptr,5+x*42,5+y*18,
          5+x*42+35,5+y*18+12,true,SwitchOn,MSClick);
      end;
```

VisualButtonPress Function TEGLUNIT

Function

Performs the pressing and releasing of a button type icon, controlled by the holding down of the left mouse button. Returns when either the user releases the left mouse button or the mouse cursor wanders off the defined mouse click area.

Declaration

VisualButtonPress(frame,mouseopt:msclickptr)

Result type

Returns true if mouse button was released while the mouse cursor overlaps with the button icon.

Remarks

This procedure may be used whenever the Frame and the Mouse Click Option is known. If the program is waiting for the user to click on one of a series of unknown icons, use CheckForMouseSelect to do an automatic Frame and Mouse click Option search.

VisualButtonPress is excellent as an entry routine for an event, since the frame and mouse click position are known.

Restrictions

This routine only works with icons that has a black fringe of two pixels wide on the right and bottom of the icon. The defined mouse click area should not include this shadow area ie. x1 and y1 is less two pixels.

See also

PressButton, CheckForMouseSelect

Example

The following example creates (8) button type icons, allowing the mouse cursor to glide over (while the buttons simulates the on/off motions). A series of beeps are sounded when the mouse button is released with the mouse cursor is on a button.

```
var x,y : word;

function SwitchOn(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word;
   begin
      Beep(1500,1,10);
      if VisualButtonPress(Frame,MouseClickPos) then
        slidebeep(500,4000,2);
      SwitchOn := 1;
```

```
   end;

PushImage(1,1,100,100);
shadowbox(1,1,100,100);

for x:=0 to 1 do
   for y:=0 to 4 do
      begin
        Putpict(stackptr^.x+6+x*42,stackptr^.y+6+y*18,
          @imageBlankBut,black);
        DefineMouseClickArea(stackptr,5+x*42,5+y*18,
          5+x*42+35,5+y*18+12,true,SwitchOn,MSClick);
      end;
```

Animation

The Animation unit provides the tools to animate a series of icons. Combined with the Icon Editor, an event can come to life.

Animation in its simplest form is the sequential display of frames. A frame in the sense of the animator is a single still image that is displayed. By linking a series of frames, animation is achieved by displaying each frame in sequence.

Using Object-Oriented Programming (OOP), the animation is as simple as declaring a object, initializing the object, then animating the object.

The description for each of the functions and procedures in this chapter is slightly different from the other chapters. Since the Animation unit is written in OOPS, the description refers to objects and methods. When referencing an object's method the object name is prefixed to the method using the dot "." . The dot is used with objects just as it is with records.

As an example:

```
VAR
  BounceIcon     : animateobject;

  BounceIcon.ResetFrame(1);
  BounceIcon.animateinit;
  BounceIcon.origin(604,wy);
  BounceIcon.animate(BounceIcon.destination(wx,wy));
```

The methods are ResetFrame, AnimateInit, Origin, Animate and Destination. The object is BounceIcon. An object's methods and data can also be accessed using WITH.

The above example may also be expressed as:

```
VAR
  BounceIcon     : animateobject;

  WITH BounceIcon DO
    BEGIN
      ResetFrame(1);
      Animateinit;
      Origin(604,wy);
```

```
      Animate(Destination(wx,wy));
    END;
```

Animation Overview

Animating a series of icons is relatively easy with the methods in the Animation unit. The hardest part is creating the series of icons and coordinating the movement differences between them.

The first step is to declare an object of AnimateObject. Here BounceIcon is declared as the object type AnimateObject.

```
  VAR BounceIcon     : AnimateObject;
```

The data within the object BounceIcon must be initialize before we can begin adding frame sequences. To initialize an object of BounceIcon, use the method Init.

```
  BounceIcon.Init;
```

The next step is to add an icon frame to the object. The method AddFrame adds an icon frame sequence to the object. The parameters are from left to right; the icon constant, defined in TEGLIcon Unit; (-15,0) the horizontal and vertical travel offset, respectively, on completion of this frame sequence; (14,37) the height and width of the icon; (10) the duration in (milliseconds) before progressing to the next sequence; (0,0) the sound in hertz, and duration; (black) the color replacement for any black pixels in the icon. In this case, black replaces black.

```
  BounceIcon.AddFrame(@imageblankbut,-15,0,14,37,10,0,0,black);
```

An object can have a number of different frame sequences. In our example, we need two sequences; a sequence for animating from the right side of the screen to the left side and a sequence for animating from the left to the right. Thus we will label the above frame as Sequence 1. The labels are arbitrary numbers ranging from 0 to 65535. However, you must use this label to switch to the appropriate sequence when the frames are animated.

```
  BounceIcon.sequence(1);
```

Use the method ResetSequence to reset the counters within the object before creating the second sequence. We then assign the second sequence the arbitrary number of 2. The only difference between this AddFrame and the last AddFrame is the horizontal travel offset. Instead of -15, the value is positive, thus adding to the x coordinate.

```
  BounceIcon.ResetSequence;
  BounceIcon.AddFrame(@imageblankbut,15,0,14,37,10,0,0,black);
  BounceIcon.sequence(2);
```

The method AnimateInit, replicates the first screen to the second screen.

```
  BounceIcon.AnimateInit;
```

Set the animation origin. In our test program, we will set the icon to the middle of the screen.

```
  BounceIcon.Origin(GetMaxx div 2,GetMaxy div 2);
```

To animate the frames, we use the method Animate. Animate displays the frames until the requested frame count is reached. Since we have only one frame to animate within each sequence, the animator will loop using the same frame until it satisfies the requested frame count.

However, since we are working with coordinates, we do not know how many frames it would take to move the icon across the screen. The method Destination will perform a test run on the sequence until one of the coordinates is satisfied and passes back a count of the frames needed to reach the destination. Thus, we can use the method Destination with the method Animate to finally animate the icon.

```
  BounceIcon.sequence(1);
  BounceIcon.Animate(BounceIcon.Destination(36,0));
```

Animating from left to right.

```
  BounceIcon.sequence(2);
  BounceIcon.Animate(BounceIcon.Destination(560,0));
```

Try experimenting with the example program. You can use the same icon to add a few more frames to each sequence. Vary the travel offsets to see the effect. However, be careful that the resulting travel distance should reach the destination, otherwise the animator will loop forever trying to reach a false destination. As well, the method Destination provides only an approximate count of frames to reach the destination. The actual destination coordinate will depend on the travel offset values on each frame added or subtracted from the origin.

Animation OOPS Methods

Origin Procedure Method — ANIMATE

Function

Sets the animated object's starting origin.

Declaration

Origin(ox,oy:word)

Remarks

Sets where the first frame will be displayed.

See also

GetOrigin, Destination

Example

```
VAR apple : AnimateObject;

apple.origin(100,100);
```

GetOrigin Procedure Method — ANIMATE

Function

Gets the animated object's current coordinates.

Declaration

GetOrigin(VAR lastox,lastoy:integer)

Remarks

Returns the current coordinate from where Animate will proceed from.

The Origins of an animated object will change depending on the travel offset defined in each animation frame.

See also

Origin, Destination

Example

```
VAR Apple       : AnimateObject;
    lastx,lasty : word;

Apple.Animate(5);
Apple.GetOrigin(lastx,lasty);
```

Destination Function Method — ANIMATE

Function

Returns a count on the number of frames that is needed for animating before the sequence gets the destination coordinates dx,dy.

Declaration

GetOrigin(VAR lastox,lastoy:integer)

Result type

word. frame count.

Remarks

Destination will return a count if either x or y coordinates of the origin is less then or greater then the destination dx,dy coordinates.

Destination is only an approximation of the number of frames required to complete the travel distance. The actual movement is dependent on each frame and its travel offsets.

See also

Origin, GetOrigin

Example

```
VAR apple       : AnimateObject;

  Apple.Animate(Apple.Destination(300,300));
```

ResetFrame Procedure Method   ANIMATE

Function

Resets a sequence to begin at any frame number.

Declaration

ResetFrame(startframe : word)

Remarks

if startframe is greater then the number of frames in the sequence, the sequence is set at the last frame.

startframe of 0 will reset the sequence back to the beginning.

See also

Sequence

Example

```
VAR Apple       : AnimateObject;

Apple.ResetFrame(0);
Apple.Animate(5);
```

Sequence Procedure Method   ANIMATE

Function

Sets the sequence pointer.

Declaration

Sequence(seqnum:word)

Remarks

seqnum is any number associated with a sequence of frames. If the sequence number does not exist, the method will assume that a new sequence will be created.

Creating a new sequence, simply records the seqnum and the start frame. So creating a sequence can occur anytime after adding the first frame. You can continue to add frames after Sequence. Use ResetSequence to clear and start a new sequence.

See also

ResetSequence, ResetFrame

Example

```
VAR Apple       : AnimateObject;

  Apple.init;
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.sequence(88);

  Apple.ResetSequence;
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.sequence(99);

  Apple.sequence(88);
  Apple.animate(5);
```

ResetSequence Procedure Method — ANIMATE

Function

Sets the internal data pointers firstframe and currentframe to nil.

Declaration

ResetSequence

Remarks

ResetSequence will reset the internal data pointers to nil. This will allow a new sequence to begin.

Restrictions

Use the method Sequence to save the data pointers, otherwise all created frames will be lost.

See also

ResetSequence, ResetFrame

Example

```
VAR apple       : AnimateObject;

  Apple.init;
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.sequence(88);

  Apple.ResetSequence;
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.addframe(@imageapple,mx,my,ht,wd,dl,hz,hzdl,color);
  Apple.sequence(99);

  Apple.sequence(88);
  Apple.animate(5);
```

AddFrame Procedure Method — ANIMATE

Function

Add a animation frame.

Declaration

AddFrame(pp:pointer; mx,my: integer; ht,wd,dy,hz, hzdy,co:word)

Remarks

AddFrame is the icon definition pointer.

mx,my is the travel offsets that are added to the origin after the icon is displayed.

ht,wd is the height and width of the icon. These parameters are used to save the background image before drawing the icon.

dy is the delay in milliseconds after displaying the image.

hz,hzdy is the frequency of the frame sound, and hzdy is the duration. If the duration of hzdy is longer then the image dy, then dy is used for the frame and the sound is left on after the frame ends.

co is the replacement color for the BLACK color pixels defined in the icon.

Restrictions

Use the method Sequence to save the data pointers, otherwise all created frames will be lost.

See also

ResetSequence, ResetFrame

Example

```
VAR apple       : AnimateObject;

  Apple.Init;
  Apple.Addframe(@imageblankbut,-15,0,14,37,10,0,0,black);
  Apple.Animate(5);
```

CurrentFrameNumber Function Method — ANIMATE

Function

Returns the current frame number.

Declaration

CurrentFrameNumber

Result type

word.

See also

ResetFrame

AnimateInit Procedure Method — ANIMATE

Function

Replicates the first active screen page to the second in preparation for animating.

Declaration

AnimateInit

See also

ResetFrame

Animate Procedure Method — ANIMATE

Function

Begins the Animation Sequence.

Declaration

Animate(numframe : word)

Remarks

numframe is the number of frames to animate. If the number of frames in a sequence is less then the requested numframe, then the sequence loops to the beginning.

Restrictions

Since animate uses two video pages, the method AnimateInit must be called to replicate the first page to the second.

See also

ResetFrame, Destination

Complete Procedure Method — ANIMATE

Function

Closes the Animation Sequence.

Declaration

Complete

Remarks

Complete toggles the sound off and resets the frame to the beginning.

Example Animation

```
{$F+}
USES Graph,crt,SoundUnt,FastGrph,TEGLUnit,Animate,TEGLIcon;

VAR BounceIcon   : AnimateObject;

BEGIN
   EGA640x350x16;

   setfillStyle(widedotfill,lightgray);
   bar(0,0,getmaxx,getmaxy);

   with BounceIcon do
      begin
        init;
        addframe(@imageblankbut,-15,-3,14,37,10,0,0,black);
        addframe(@imageblankbut,-15,3,14,37,10,0,0,black);
        addframe(@imageblankbut,-15,3,14,37,10,0,0,black);
        addframe(@imageblankbut,-15,-3,14,37,10,0,0,black);
        sequence(1);

        ResetSequence;
        addframe(@imageblankbut,15,-3,14,37,10,0,0,black);
        addframe(@imageblankbut,15,3,14,37,10,0,0,black);
        addframe(@imageblankbut,15,3,14,37,10,0,0,black);
        addframe(@imageblankbut,15,-3,14,37,10,0,0,black);
        sequence(2);

        Animateinit;
        Origin(getmaxx div 2,getmaxy div 2);

        ClearKeyBoardBuf;
        while not keypressed do
           begin
              sequence(1);
              ResetFrame(0);
              Animate(Destination(36,0));
              Beep(1500,1,1);

              sequence(2);
              ResetFrame(0);
              Animate(Destination(560,0));
              Beep(1500,1,1);
           end;
      end;

   ABORT('BYE...');
END.
```

Chapter 10 - Animation

Chapter 11 - Writing Text

Writing Text

TEGL Windows Toolkit provides the tools to write to the screen using proportional bit-mapped fonts. Fonts may be as small as 5 pixels high and 3 pixels wide or as large as 24 pixels high and 8 pixels wide.

Both BGI vector fonts and TEGL bit-mapped fonts may be used together. Like TP's OutTextXY procedure, TEGLOutTextXY is affected by the SetTextJustify procedure. To turn off the Proportional print, use the procedure SetProportional(false).

TEGLWrt Variables

Bit-mapped Fonts

There are 25 bit-mapped fonts available in the TEGLWrt unit. They are:

FONT09, FONT14, COUNTDWN, OENGLISH, SCRIPT, OCR, FRAKTUR, ITALIC, GEORGIAN, APLS7, PC9, GAELIC, LITALIC, PC24, PC3270, M3270, EGA09, FUTURE, BROADWAY, SCRIPT2, LCDFONT, LIGHT14, BRDWX19, SANSX19, WNDWX19, LIGHT9.

To select a font, just pass the address to SetTEGLFont. i.e. SetTEGLFont(@COUNTDN).

Creating Your Own Bit-mapped Fonts

You can create and add your own fonts by modifying the assembler files then assembling the new font to to an object file. Each bit in a byte represents a pixel of the font.

The format of a TEGL font is:

```
    1 byte header - indicating the height of the font.

    Each character is:
    1 byte  - proportional font width
    n bytes - defined by the 1 byte header
```

TEGLWrt Functions and Procedures

OutTEGLTextXY Procedure   FASTGRPH

Function

Writes mystr to the graphics screen at x,y.

Declaration

OutTEGLTextXY(x,y : integer; mystr : string)

Remarks

OutTEGLTextXY is affected by the justification settings set by SetTextJustify and color by SetColor.

x,y is the coordinates of the graphic screen.

mystr is the text string for output.

FontTable is a global variable which is used to set the pointer to an internal font table.

See also

TEGLWrtChar

Example

```
  SetTextJustify(CenterText,CenterText);
  SetColor(green);
  SetTEGLFont(@Script);
  OutTEGLTextXY(100,100,'TEGL Systems Corporation');
```

TEGLTextWidth Function FASTGRPH

Function

Returns the proportional width of mystr.

Declaration

TEGLTextWidth(mystr : string)

Result type

integer size of mystr.

Remarks

TEGLTextWidth will scan and total the exact number of pixels mystr will occupy.

Restrictions

Any unprintable characters will not be included in the final size.

See also

TEGLCharWidth Function FASTGRPH

Function

Returns the proportional width of a character.

Declaration

TEGLCharWidth(c : word)

Result type

Word.

Remarks

c is the ordinal value of the character.

TEGLCharWidth will return a value based on the currently selected font.

Restrictions

Characters outside the 28-126 ascii code will return a invalid size.

See also

TEGLTextWidth, TEGLCharHeight

TEGLCharHeight Function FASTGRPH

Function

Returns the height of the proportional font.

Declaration

TEGLCharHeight

Result type

Word.

Remarks

TEGLCharHeight will return to the first byte in the font table which is the height of the current font.

See also

TEGLTextWidth, TEGLCharWidth

TEGLWrtChar Procedure FASTGRPH

Function

Writes a single character to the graphics screen.

Declaration

TEGLWrtChar (c,x,y,color:word)

Remarks

x,y specifies the coordinates for writing the character.

c is the ascii code of the character. Valid character range is 28-126.

color is color of the output character.

See also

TEGLOutTextXY

SetProportional Procedure FASTGRPH

Function

Switch Proportional font on or off.

Declaration

SetProportional(onoff:boolean)

Remarks

Default is proportional font on TRUE. If proportional font is off FALSE, the spacing is 8 bits.

SetTEGLFont Procedure FASTGRPH

Function

Sets the font to use in subsequent calls to OutTEGLTextXY.

Declaration

SetTEGLFont(P : Pointer);

Remarks

This procedure simply sets the FontTable variable to the address in P.

UnderLineChar Function FASTGRPH

Function

Returns the character with the high bit set.

Declaration

UnderLineChar(c : Char): Char;

Remarks

OutTEGLTextXY detects characters with the high bit set and underlines them.

Restrictions

Underline does not work with TEGLWrtChar.

Underline does not work on characters with decenders.

Showing ALL Fonts FONTTEST.PAS

The TEGLSam.PAS demonstration program uses the FontTest unit to display all available fonts, or, individual fonts by selecting from a menu.

FontName Function FONTTEST

Function

Returns the name of a font.

Declaration

FontName(fontnum:word);

Result type

string.

Remarks

FontName is used to build the menu for selective display of fonts.

See also

ShowOneFont, ShowFonts

ShowOneFont Event FONTTEST

Function

An Event that displays a font based on MouseClickPos^.ClickNumber.

Remarks

FontName is used to build the menu for selective display of fonts. The entries are positional, thereby each menu MouseClickPos selection corresponds to a fontnumber.

See also

FontName, ShowFonts

ShowFonts Event | FONTTEST

Function

A TEGL Event that displays all fonts.

Declaration

```
ShowFonts(Frame:imagestkptr; Ms: MsClickPtr) : word;
```

Remarks

A TEGL Event that displays all the available fonts and their respective names.

See also

FontName, ShowOneFont

Event Library

Although we call it a library, the Event's covered here span over several units.

The event library contains events that may be used immediately in programming an application.

The File Selector

The file selector SelectaFile provides a dialogue event, that displays the files of a directory and lets the user select one of the existing files or enter a new file name.

The file selector dialogue box allows the user to choose any displayed file either by clicking on the file name and then clicking on the OK button or by clicking on the selection area and typing in the filename.

To change directories, position the mouse cursor at a directory filename and click or click at the bar at the top of the file selector window and type in the directory path.

SelectaFile will return the full file name, including the directory prefix, for the file selected. If the Cancel button was clicked or no file was selected, the file name returned will be an empty string.

SelectaFile function SELECTFL

Function

Provides a file selection dialogue that allows a user to choose or create a new filename.

Declaration

```
Selectafile(x,y:word; var path,fileselected:
string)
```

Result type

boolean. True if a file was selected. False if no file was selected or the mouse clicked on the cancel button.

Remarks

x,y is the coordinates where the file selection dialogue will be displayed.

path is the original directory path specification. Use a global string variable to retain the last directory path.

fileselected will contain the selected path and filename, if the function returns True.

Example

```
function FileSelect(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word;
   var x,y,x1,y1       : word;
       IFS             : imagestkptr;
       selected        : boolean;
       selectedfile    : string;
   begin
      selected := selectafile(100,100,path,selectedfile);

      hidemouse;
      x  := 10;
      y  := 60;
      x1 := x+500;
      y1 := y+100;

      PushImage(x,y,x1,y1);
      IFS := stackptr;
      shadowbox(x,y,x1,y1);
      setcolor(black);

      if not selected then
         outTEGLtextxy(x+5,y+3,'No file were selected.')
      else
        begin
           outTEGLtextxy(x+5,y+3,'The file selected is:');
           FONTTABLE := @FONT09;
           outTEGLtextxy(x+5,y+17,selectedfile);
           FONTTABLE := @font14;
        end;

      Putpict(x+280,y+75,@imageok,black);
      DefineMouseClickArea(IFS,280,75,280+35,75+12,true,
        nilunitproc,MSCLICK);
      setmouseposition(x+290,y+85);
      showmouse;

      while CheckforMouseSelect(IFS)=nil do;

      hidemouse;
      dropstackimage(ifs);
      showmouse;
      fileselectionoption := 1;
   end;
```

String Editing Dialog

The EditString procedure provides a facility for getting text input from the user. The file selector uses this routine to get a new filename.

EditString Procedure — SELECTFL

Function

Provides string input facility.

Declaration

```
EditString(fs:imagestkptr; x,y,maxlen : word;
var textstr : string)
```

Remarks

fs is of the type imagestkptr, created by pushimage.

x,y is the relative coordinates from the upper left of fs where a blinking vertical bar and text input will be displayed.

maxlen is the number of maximum number of input characters.

textstr is the user input string.

Restrictions

String editing should be on the topmost window.

Example

```
VAR mystring;

  pushimage(100,100,150,150);
  FONTTABLE := @FONT14;
  Editstring(stackptr,5,5,12,mystring);
```

Mouse Sensitivity Dialogue Window

The mouse sensitivity dialogue box allows the user to change the horizontal, vertical and threshold settings of the mouse. The dialogue box consists of radio type buttons that can adjust the numeric counters.

SetMouseSense Procedure SENSEMS

Function

Provides a mouse sensitivity dialogue window that allows the user to change the sensitivity setting of the mouse.

Declaration

SetMouseSense(x,y:word)

Remarks

x,y is the coordinates where the SetMouseSense dialogue will be displayed.

Restrictions

The dialogue does not check if the mouse is present.

Example

```
function AskMouseSense(Frame:imagestkptr;
           MouseClickPos: msclickptr) : word;
   begin
      SetMouseSense(160,75);
      AskMouseSense := 1;
   end;
```

Bells & Whistles, Sound Unit

The AskSoundSense dialogue window allows the user to change the duration of the beeps and whistle settings of the sound unit. The dialogue box consists of radio type buttons that can adjust the numeric counters.

AskSoundSense Event SOUNDUNT

Function

A sound duration dialogue event

Remarks

An event that displays a dialogue box that permits the user to set the sound duration for beeps and whistles.

Beep Procedure SOUNDUNT

Function

Toggles the sound on for a specific tone and duration for n times.

Declaration

beep(tone,n,duration:integer)

Remarks

tone specifies the frequency of the emitted sound in hertz.

n specifies the number of times the sound it toggle on and off.

duration specifies the length in milliseconds of the sound.

See also

SlideBeep, SoundSwitch

Example

```
beep(1000,3,100);
```

SlideBeep Procedure SOUNDUNT

Function

Performs a sliding type of sound. Whistle type.

Declaration

slidebeep(tone1,tone2,n:integer)

Remarks

tone1 specifies the initial frequency of the emitted sound in hertz. tone2 specifies the second frequency from which tone1 steps towards.

n specifies the number of times the slide beep occurs.

See also

Beep, SoundSwitch

Example

```
slidebeep(1000,2000,2);
```

SoundSwitch Procedure SOUNDUNT

Function

Switches the sound function on/off.

Declaration

SoundSwitch(OnOff:boolean)

Remarks

OnOff switches the sound on True or off False.

See also

Beep, SlideBeep

Chapter 13 - Virtual Memory Manager

Virtual Memory Manager

Graphical images, by their nature, require a tremendous amount of memory to store and manipulate. Combine this with the DOS limitation of 640k, writing applications using a graphical environment can be limiting.

Virtual Memory is a concept by which less expensive mass storage devices (ie. hard disk) may be used as though it were an extension of memory. Then memory is only limited by the size of the hard disk.

The TEGL virtual memory manager may be used within your application program independent of its use within the TEGL window manager.

In this chapter, we provide technical information for advanced programmers. We'll cover topics such as the Virtual Memory Manager, Turbo Pascal's heap manager, Expanded Memory Manager, calling conventions, and more.

The Turbo Pascal Heap manager is covered in greater detail in the Turbo Pascal Reference Guide, Chapter 15, Inside Turbo Pascal.

Heap Management

With Window Management routines, the memory requirement is unknown. If we were to attempt to ensure that memory is available for every window that is created within the program, we would have an unwieldy and unjustifiably large program. In actual fact, any modest application would require much more memory than is available.

Rather then attempting to reserve a fixed amount of memory space, which places a limitation on the program, the heap provides the facility of allocating memory dynamically. The heap permits us to allocate memory only when it is required and to release the memory when the task is completed.

The Turbo Pascal Heap Manager

In Turbo Pascal the heap is all the remaining memory that is left when a program is executed.

Memory is allocated from the heap starting with the lowest part of the heap growing upwards. The bottom of the heap is stored in the variable HeapOrg. Each time a block of memory is allocated on the heap (via New or GetMem), the heap manage moves HeapPtr upward by the size of the requested block.

The top of the heap, or the maximum size of the heap is controlled by the variable FreePtr. It does not point directly at the maximum top, rather it points at the start of the free pointer chain.

The free pointer chain grows downward as memory blocks are freed. Adjacent memory blocks are always combined to form larger blocks.

Chapter 13 - Virtual Memory Manager

The maximum size of a single block of memory, using Turbo Pascal's heap manager, is 65519 bytes.

The TEGL Heap Manager
The TEGL Heap Manager allows us to allocate memory blocks that are greater than 64k. A full EGA screen image (640x350 -16 colors) is approximately 109k.

When a memory request is made to the TEGL Heap Manager, the manager will attempt to allocate memory between HeapPtr and FreePtr first, before attempting to find space on the free space list.

Turbo Pascal Heap manager differs from the TEGL Heap Manager in that TP will search through the free space chain and reuses the first available memory block that can accommodate the request.

If memory allocation is less then 64k, use Turbo Pascal's GETMem and Freemem.

Use the TEGL Heap Manager sparingly, as this will reduce the amount of memory managed by the virtual memory handler (see Resolving Fragments in this chapter).

The TEGL Heap Error Function
The HugeHeapError variable allows you to install a heap error function, which gets called whenever the TEGL heap manager cannot complete an allocation request. HugeHeapError is a pointer that points to a function with the following header:

```
    {$F+} {sh function} ReturnHeapError(size: longint) : word;  {$F-}
```

The TEGL heap error function is installed by assigning its address to the HugeHeapError variable:

```
    HugeHeapError := ReturnHeapError;
```

The TEGL heap error function gets called whenever a call to TEGLGetMem cannot complete the request. The Size parameter contains the size of the block that could not be allocated, and the TEGL heap error function should attempt to free a block of at least that size.

Depending on its success, the TEGL heap error function may return a 1 or 2. A return of 2 indicates success and causes a retry (which could also cause another call to the TEGL heap error function). A return of any other value

will cause TEGLGetMem to return a nil pointer.

The standard TEGL heap error function always returns a 1, causing TEGLGetMem to return a Nil pointer.

TEGLUnit sets the heap error function to point to the virtual memory manager. Don't use the heap error function if you are using TEGLUnit, the virtual memory handler depends on this function to know when its time to start paging out window buffers.

The TEGL Heap Manager Functions

TEGLGetMem Procedure VIRTMEM

Function

Returns a pointer to a memory block of the specified size.

Declaration

TEGLGetMem(var Pt: pointer; size: LongInt);

Remarks

Pt is a pointer variable of any pointer type. Size is a longint specifying the size, in bytes, of the memory block to allocate.

If there isn't enough free space on the heap to allocate the memory block, Pt is set to nil. A user defined run-time error procedure can be used to intercept any heap errors (see HugeHeapError).

TEGLGetMem is compatible with Turbo Pascal's Memory manager and may be used interchangeably.

Restrictions

There are actually no restrictions on the size of the largest block that can be allocated, however, DOS limits you to the remaining memory after the program is loaded.

See also

TEGLFreeMem

Example

Allocates and frees a 128k buffer.

```
Uses VirtMem;

Var buffer : pointer;

begin
```

```
  TEGLGetMem(buffer,131072);
  TEGLFreeMem(buffer,131072);
end.
```

TEGLFreeMem Procedure VIRTMEM

Function

Frees a memory block and returns the memory back to the heap manager.

Declaration

TEGLFreeMem(var Pt: pointer; size: LongInt);

Remarks

Pt is a pointer variable of any pointer type that was previously assigned by the GetMem or TEGLGetMem procedure. Size is a longint specifying the size of the memory block, in bytes, to be freed; it must be exactly the same number of bytes previously allocated to that memory block by GetMem or TEGLGetMem. TEGLFreeMem returns the memory region to the heap.

TEGLFreeMem is compatible with Turbo Pascal's Memory manager and may be used interchangeably with GetMem or TEGLGetMem (see Resolving Fragments for restrictions).

Restrictions

You can use TEGLFreeMem to free memory blocks that were allocated by Turbo Pascal's Getmem. However, TEGLFreeMem organizes the free space pointer chain in a sorted order in order to minimize any free space fragmentation. If ReserveHugeMinimum is used to partition the heap, use the respective counterparts to allocated and free the memory (GetMem/FreeMem, TEGLGetMem/TEGLFreeMem).

See also

TEGLGetMem

Expanded Memory Manager (EMM)

The Expanded Memory Manager is a device driver that controls and manages expanded memory and application programs that use expanded memory.

Expanded memory is memory beyond DOS's 640K-byte limit. The Expanded Memory specification (EMS) supports up to 32M bytes of expanded memory.

Chapter 13 - Virtual Memory Manager

Because the 8086, 8088, and 80286 (in real mode) microprocessors can physically address only 1M byte of memory, they access expanded memory through a window in their physical address range.

This is similar to a book, where pages within the book can retain data. However, just like a book, if you wish to retrieve the data, you must supply the page number. As well, when you first create the book (returning a handle) the initial number of pages must be specified. If you require more pages after the initial allocation, a new book must be created (Version 3.2 EMS did not provide a function that allows you to expand the initial allocation with the same handle).

There are approximately 30 EMS functions calls available with EMS Version 4.0; as documented in the specification produced jointly by Lotus Development Corporation, Intel Corporation, and Microsoft Corporation. A copy of this documentation (Part number 300275-005) October, 1987, can be obtained from Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

However, EMM Version 3.2 is still widely used as the driver on most systems, and therefore we are limited in terms of compatibility, to the number of functions that may be used.

Expanded Memory Functions

EmmInstalled function — VIRTMEM

Function

Returns an installed status on the Expanded Memory Manager.

Declaration

EmmInstalled

Result type

Returns a boolean status of true, if an EMM driver is installed on the system, false if not installed.

Remarks

This function uses the address that is found in the Int 67H vector to inspect the device header of the presumed EMM. If the EMM is present, the name field at offset 0AH of the device header will contain the string EMMXXXX0.

EMSPagesAvailable function — VIRTMEM

Function

Obtains the total number of expanded memory pages present in the systems, and the number of those pages that are not already allocated.

Declaration

EMSPagesAvailable (Var TotalEMSPages, PagesAvailable: Word)

Result type

Returns a return code of 0 if EMM software is successful. A return code other then 0 indicates a possible error in the EMM software or a memory hardware error.

Remarks

This function may be used to determine the number of pages available before allocating EMS pages.

AllocateExpandedMemoryPages function — VIRTMEM

Function

Allocates the requested number of pages (16k per page) and returns a handle that is used to reference the allocated pages.

Declaration

AllocateExpandedMemoryPages(PagesNeeded:Word; Var Handle:Word)

Result type

Returns a return code of 0 if EMM software is successful. A return code of $88 indicates that the requested sh PagesNeeded is greater then the number of pages that is currently available in the system.

See also

MapExpandedMemoryPages, GetPageFrameBaseAddress, DeallocateExpandedMemoryPages

MapExpandedMemoryPages function — VIRTMEM

Function

Maps one of the logical pages of expanded memory assigned to a handle onto one of the four physical pages within the EMM's page frame.

Declaration

MapExpandedMemoryPages(Handle,LogicalPage, PhysicalPage: Word)

Result type

Returns a return code of 0 if EMM software is successful. A return code of $8A indicates that the logical page requested to be mapped is outside the range of pages that is currently assigned to the handle.

Remarks

A logical page is one page from the range of pages that were allocated through the sh AllocateExpandedMemoryPages procedure. The logical-page number must be in the range {0_._._._n_-_1}}, where {it n} is the number of logical pages previously allocated.

A physical page is one of four 16k byte pages, in the range of 0-3, that may viewed as the window to the expanded memory. Use sh GetPageFrameBaseAddress to obtain the segment address to the physical window.

See also

AllocateExpandedMemoryPages, GetPageFrameBaseAddress, DeallocateExpandedMemoryPages

---

GetPageFrameBaseAddress function VIRTMEM

Function

Returns the segment address of the page frame used by the Expanded Memory Manager.

Declaration

GetPageFrameBaseAddress(Var PageFrameAddress: Word)

Result type

Returns a return code of 0 if EMM software is successful. A return code other then 0 indicates a possible error in the EMM software or a memory hardware

error.

Remarks

This is only the segment address of the physical page frame. Use offsets of $0000 for physical page 0, offset of $4000 for page 1, offset of $8000 for page 2 and offset of $C000 for page 3.

See also

AllocateExpandedMemoryPages, MapExpandedMemoryPages, DeallocateExpandedMemoryPages

---

DeallocateExpandedMemoryPages function — VIRTMEM

Function

Deallocates (releases) the pages of expanded memory currently assigned to a handle.

Declaration

DeallocateExpandedMemoryPages (Handle: Word)

Result type

Returns a return code of 0 if EMM software is successful.

Remarks

This function notifies the Expanded Memory Manager that the application will not be making further use of the allocated expanded memory pages. This function would typically be called by a program just before performing an exit.

See also

AllocateExpandedMemoryPages, MapExpandedMemoryPages, GetPageFrameBaseAddress.

GetVersionNumber function — VIRTMEM

Function

Returns the EMM Version Number in a string format. A handle.

Declaration

GetVersionNumber(Var VersionString: string)

Result type

Returns a return code of 0 if EMM software is successful. A return code other then 0 indicates a possible error in the EMM software or a memory hardware error.

Remarks

This function returns a EMM Version Number that may be used to check if the installed EMM will support the requested functions. However since Version 4.00 of the expanded memory specification is downward compatible with Version 3.2, this function is only useful as information.

GetHandleCountUsed function VIRTMEM

Function

Returns the number of total handles used by all applications. a handle.

Declaration

GetHandleCountUsed (var NumberOfHandles: Word)

Result type

Returns a return code of 0 if EMM software is successful. A return code other then 0 indicates a possible error in the EMM software or a memory hardware error.

Remarks

The number of available handles depends on the parameters used to start up the EMM driver, as well as the number of handles in use by other resident or multitasking software. The upper limit in Version 4.00 is 255 handles with a lower limit of 32. If the returned number of handles is zero, the EMM is idle and none of the expanded memory is in use.

GetPagesOwnedByHandle function VIRTMEM

Function

Returns the number of expanded memory pages allocated to a specific EMM handle.

Declaration

GetPagesOwnedByHandle (Handle: Word; Var

PagesOwned:word):

Result type

Returns a return code of 0 if EMM software is successful.

Remarks

An EMM handle never has zero pages of memory allocated to it.

Expanded Memory Test Program

```
program EmsTest;

uses VirtMem;

Var
  EmmHandle,
  PageFrameBaseAddress,
  PagesNeeded,
  PhysicalPage,
  LogicalPage,
  Offset,
  ErrorCode,
  PagesEMSAvailable,
  TotalHandleCount,
  PagesOwned,
  TotalEMSPages,
  AvailableEMSPages: Word;

  VersionNumber,
  PagesNumberString: string;

  Verify: Boolean;

  DataPtr : pointer;


FUNCTION HexString(I : word) : string;
   FUNCTION HexByte(B : byte) : string;
      const HexDigit : ARRAY[0..15] OF Char = '0123456789ABCDEF';
      BEGIN
       HexByte := HexDigit[B SHR 4]+HexDigit[B AND $F];
      END;
   BEGIN
     HexString := HexByte(Hi(I))+HexByte(Lo(I));
   END;

Procedure Error(ErrorMessage: string; ErrorNumber: Word);
   Begin
```

```
      Writeln(ErrorMessage);
      Writeln('  ErrorNumber = ',HexString(ErrorNumber) );
      Writeln('EMS test program aborting.');
      Halt(1);
   end;

Begin
  { Determine if the Expanded Memory Manager is installed }
  If not (EmmInstalled) then
    Error('The LIM Expanded Memory Manager is not installed.',255);

  { Get Version number}
  ErrorCode:= GetVersionNumber(VersionNumber);
  If ErrorCode<>0 then
    Error('Error trying to get the EMS version number ',Errorcode);
  Writeln('LIM Expanded Memory Manager, version ',VersionNumber);
  Writeln;

  { Get the expanded memory page frame address }
  ErrorCode:= GetPageFrameBaseAddress(PageFrameBaseAddress);
  If ErrorCode<>0 then
    Error('Error trying to get the base Page Frame Address.',ErrorCode);
  Writeln('The base address of the EMS page frame is - '+
         HexString(PageFrameBaseAddress) );
  Writeln;

  { Get Available pages. }
  ErrorCode:= EMSPagesAvailable(TotalEMSPages,AvailableEMSPages);
  If ErrorCode<>0 then
    Error('Error in determining available EMS pages.',Errorcode);
  Writeln('There are ',TotalEMSPages,' pages present in this system.');
  Writeln('  ',AvailableEMSPages,' of those pages are available.');
  Writeln;

  { Get Handle Count }
  ErrorCode:= GetHandleCountUsed(TotalHandleCount);
  If ErrorCode<>0 then
    Error('Error in getting the Handle Count Used.',ErrorCode);

  { Determine if there are enough pages for this application.}
  PagesNeeded:=1;
  If PagesNeeded>AvailableEMSPages then
     Begin
       Str(PagesNeeded,PagesNumberString);
       Error('We need '+PagesNumberString+' EMS pages. ' +
            'There are not that many available.',ErrorCode);
     end;

  { Allocate expanded memory pages for our usage }
  ErrorCode:= AllocateExpandedMemoryPages(PagesNeeded,EmmHandle);
  Str(PagesNeeded,PagesNumberString);
  If ErrorCode<>0 then
```

```
  Error('Error in allocating '+PagesNumberString+
       ' pages for usage.',ErrorCode);
Writeln(PagesNeeded,' EMS page(s) allocated for the EMS test program.');
Writeln;

{ Map in the required logical pages to the physical pages }
LogicalPage :=0;
PhysicalPage:=0;
ErrorCode:=MapExpandedMemoryPages(EmmHandle,LogicalPage,PhysicalPage);
If ErrorCode<>0 then
  Error('Error in mapping logical pages onto physical pages.',ErrorCode);
Writeln('Logical Page ',LogicalPage,
       ' successfully mapped onto Physical Page ',PhysicalPage);
Writeln;

{ Get the number of pages for our handle }
ErrorCode:= GetPagesOwnedByHandle(EmmHandle,PagesOwned);
If ErrorCode<>0 then
  Error('Error in getting number of pages Owned by handle.',ErrorCode);
Writeln('The Total Handle Count is ',TotalHandleCount,
       ' and the number of Pages owned is ',PagesOwned,'.');
Writeln;


{ Write a test pattern to expanded memory }
For Offset:=0 to 16382 do
  Mem[PageFrameBaseAddress:Offset]:=Offset mod 256;

{ Make sure that what is in EMS memory is what we just wrote }
Writeln('Testing EMS memory.');

Offset:=1;
Verify:=True;
while (Offset<=16382) and (Verify=True) do
   Begin
     If Mem[PageFrameBaseAddress:Offset]<>Offset mod 256 then
        Verify:=False;
     Offset:=Succ(Offset);
   end;

{ If it isn't report the error }
If not Verify then
  Error('What was written to EMS memory was not found during '+
       'memory verification  test.',0);
Writeln('EMS memory test successful.');
Writeln;


{ Return the expanded memory pages back to the EMS memory pool }
ErrorCode:=DeallocateExpandedMemoryPages(EmmHandle);
If ErrorCode<>0 then
  Error('EMS test program was unable to deallocate '+
```

```
         'the EMS pages in use.',ErrorCode);
  Writeln(PagesNeeded,' page(s) deallocated.');

  Writeln;
  Writeln('EMS test program completed.');
end.
```

A RAM Disk Driver

Expanded Memory (EMS), in its architecture of multiple pages, is limited in its use as a direct access heap without complex programming. However, one of the simplest ways to take advantage of EMS, is to create a EMS ram disk.

The following EMS RAM Disk functions provides the basics for storing and retrieving a file from EMS memory.

EMSOpen function — VIRTMEM

Function

Opens an EMS Ram Disk file.

Declaration

EMSOpen(MinimumPages:word)

Result type

EMSOpen returns a variable of type EMSFile.

Remarks

EMSFile is predeclared as follows:

```
type
  EMSBlockPtr = ^EMSBlock;
  EMSBlock    = Record
           nextblockptr  : EMSblockPtr;
           Handle        : word;        {Multiple handles}
           EMSPage       : word;        {Pages allocated}
        end;

  EMSFile     = ^EMSFileRec;
  EMSFileRec  = Record
           PageOffset    : word;     {current offset within page}
           BaseAddress   : word;
           EMSPosition   : longint;
           TotalPages    : word;     {Total number of 16k pages}
           RootBlkPtr    : EMSBlockPtr;
```

```
end;
```

The BaseAddress and PageOffset forms the pointer to the physical expanded memory page. The EMSPosition field is the current RAM disk file position. TotalPages is the total number of expanded memory pages allocated for this EMS Ram file. The RootBlkPtr points to the first EMS Block pointer.

The MinimumPages parameter specifies the initial allocation, however if more pages are required, as you write to the EMS Ram file, pages are automatically allocated as needed. Additional EMS handles and Pages information are stored in separate EMS Block records and are chained together.

EMS_Status will return a 0 if the EMS ram file is allocated successfully; otherwise, it will return a nonzero error code.

See also

EMSClose

EMSSeek procedure VIRTMEM

Function

Moves the current position of an EMS RAM file to a specified byte component.

Declaration

EMSSeek(var EMSRamFile:EMSFile; Position: longint)

Remarks

EMSRamFile is the record type returned by EMSOpen, and Position is an expression of type longint. The current EMS Ram file position is moved to the offset Position. In order to expand the expanded memory pages allocated, it is possible to EMSSeek any size beyond the last byte; thus EMSSeek(myramfile, 98304) will automatically allocate, if required, a total of 6 pages.

EMS_Status will return a 0 if the operation was successful; otherwise, it will return a nonzero error code.

Restrictions

EMS Ram file must be open.

See also

EMSBlockWrite, EMSBlockRead, EMSOpen, EMSClose

EMSBlockWrite procedure VIRTMEM

Function

Writes the information pointed to by the Buffer pointer to the EMS Ram file.

Declaration

EMSBlockWrite(var EMSRamFile:EMSFile; buffer: pointer; bytestowrite:longint)

Remarks

EMSRamFile is the record type returned by sh EMSOpen, Buffer is any pointer type, and Bytestowrite is an expression of type longint.

EMSBlockWrite writes bytestowrite bytes to the EMSRamFile. Bytestowrite may be greater than (64k). EMSBlockWrite will automatically allocate additional EMS Memory pages if the current EMS Ram file position plus Bytestowrite exceeds the currently allocated expanded memory pages.

The current EMS Ram file position is advanced by Bytestowrite on completion of EMSBlockWrite.

EMS_Status will return a 0 if the operation was successful; otherwise, it will return a nonzero error code.

Restrictions

EMS Ram file must be open.

See also

EMSSeek, EMSBlockRead, EMSOpen, EMSClose

EMSBlockRead procedure VIRTMEM

Function

Reads from the EMS Ram file to memory pointed to by the

buffer pointer.

Declaration

EMSBlockRead(var EMSRamFile:EMSFile; buffer: pointer; bytestoread:longint)

Remarks

EMSRamFile is the record type returned by sh EMSOpen, Buffer is any pointer type, and Bytestoread is an expression of type longint.

EMSBlockRead reads bytestoread bytes to the memory area pointed to by Buffer. Bytestoread may be greater than (64k). EMSBlockRead will read past the end of Ram file and automatically allocate additional EMS Memory pages if the current EMS Ram file position plus Bytestoread exceeds the currently allocated expanded memory pages.

The current EMS Ram file position is advanced by Bytestoread on completion of EMSBlockRead.

EMS_Status will return a 0 if the operation was successful; otherwise, it will return a nonzero error code.

Restrictions

EMS Ram file must be open.

See also

EMSBlockWrite, EMSSeek, EMSOpen, EMSClose

EMSClose procedure VIRTMEM

Function

Close an Open EMS Ram file.

Declaration

EMSClose(var EMSRamFile:EMSFile)

Remarks

EMSRamFile is the record type returned by sh EMSOpen.

EMS_Status will return a 0 if the operation was successful; otherwise, it will return a nonzero error code.

See also

EMSOpen

Virtual Disk Heap

A virtual Disk Heap allows you to simulate a heap using a sequential file. Allocating and freeing space within the Virtual Disk Heap are automatically maintained, with all the flexibility of a real memory heap manager and the unlimited space of a hard disk. The virtual Disk Heap manager has the ability to reuse free space, as well as merging adjacent free space fragments.

In addition the virtual disk heap (disk mode) can be used as a simple graphical image database manager. The stored images may be retrieved later by referring to a unique signature that you provide.

VDskOpenHeapFile function — VIRTMEM

Function

Opens a heap file on disk.

Declaration

VDskOpenHeapFile(VDskFileName : string; VDskAttribute:word)

Result type

VDskOpenHeapFile returns a variable of type VDskFile.

Remarks

VDskFileName is a string type expression that contains the name of heap file and VDskAttribute is the attribute that is associated with the file. The following VDskAttribute constants are declared:

```
CONST
  VDskReadWrite = 1;
  VDskTemporary = 2;
```

VDskOpenHeapFile will create a new file if the file does not exist. If VDskReadWrite is specified, the file is not erased when the file is closed. if VDskAttribute is set to VDskTemporary, the file is automatically erased when the file is closed.

VDSKFile is declared as follows:

```
type
  VDskFreePtr         = ^VDskFreeRecord;
  VDskFreeRecord      = Record
```

```
               NextVDskFree : VDskFreePtr;
               StartBlock   : longint;
               EndBlock     : longint;
               Signature    : Signate;
               BlockFree    : boolean;
            end;

  VDskFile            = ^VDskFileRecord;
  VDskFileRecord      = Record
               VDskFreePtrChain : VDskFreePtr;
               VDskTopOfFile    : longint;
               VDskAttribute    : word;
               Case EMSType : boolean of
                  false : (VDskHeapFile: File);
                  true  : (VEMSHeapFile: EMSFile);
            end;
```

VDskFreePtrChain maintains a complete list of all blocks that are allocated and freed. Information regarding each block are stored in a chain of VDskFreeRecord. The VDskTopOfFile is the position of the end of the heap file. If there are no free space fragments before the end of the heap file to satisfy the requested block size, space is allocated starting at VDskTopOfFile. VDskAttribute is the passed parameter when the file was opened. The EMSType sets the variant portion to either disk or EMS memory.

StartBlock and EndBlock is the starting and ending address of the allocated or freed block, respectively. Signature is a unique type of a 4 character string that can be used as a search string to locate an address of a block. Blockfree indicates whether the block is allocated or free.

VDSKStatus will return a 0 if the operation was successful; otherwise, it will return a nonzero error code.

See also

VEMSOpenHeapFile, VDskCloseHeapFile

VEMSOpenHeapFile function VIRTMEM

Function

Opens a heap file in EMS.

Declaration

VEMSOpenHeapFile

Result type

VEMSOpenHeapFile returns a variable of type VDskFile.

Remarks

VEMSOpenHeapFile creates the same structure as VDskOpenHeapFile, with the EMSType set to EMS memory.

VDSKStatus will return a 0 if the EMS operation was successful; otherwise, it will return a nonzero error code.

See also

VDSKOpenHeapFile, VDskCloseHeapFile

VDSKGetMem function — VIRTMEM

Function

Allocates a block within the virtual heap memory and returns a virtual heap address.

Declaration

VDskGetMem(var VDskPacket:VDskFile; HeapSize: longint; signature:Signate)

Result type

VDSKGetMem returns a virtual heap address of longint.

Remarks

VDSKStatus will return a 0 if the virtual heap allocation was successfull; otherwise, it will return a nonzero error code.

Restrictions

The Virtual Heap memory must be opened.

See also

VDSKFreeMem, VDskWriteHeapData, VDskReadHeapData

VDSKFreeMem procedure — VIRTMEM

Function

Frees the virtual heap memory pointed to by the VDskHeapPtr.

Declaration

VDskFreeMem(var VDskPacket:VDskFile; VDskHeapPtr: longint)

Remarks

VDskPacket is the record type returned by VEMSOpenHeapFile or VDskOpenHeapFile. The VDskHeapPtr must be the virtual disk pointer from VDskGetMem.

VDSKStatus will return a 0 if the virtual heap de-allocation was successful; otherwise, it will return a nonzero error code.

Restrictions

The Virtual Heap memory must be opened.

See also

VDSKGetMem, VDskWriteHeapData, VDskReadHeapData

VDSKWriteHeapData procedure — VIRTMEM

Function

Writes the data from memory pointed to by the DataPtr to an allocated virtual heap memory VDskHeapPtr.

Declaration

VDskWriteHeapData(var VDskPacket:VDskFile; DataPtr:pointer; VDskHeapPtr:longint)

Remarks

VDskPacket is the record type returned by VEMSOpenHeapFile or VDskOpenHeapFile. The DataPtr is of a pointer type that points to a memory buffer that will be written out to the virtual heap. The VDskHeapPtr must be the virtual heap pointer created from VDskGetMem.

VDSKStatus will return a 0 if writing to the virtual heap was successful; otherwise, it will return a nonzero error code.

Restrictions

The Virtual Heap memory must be opened.

See also

VDSKGetMem, VDskFreeMem, VDskReadHeapData

VDSKReadHeapData procedure VIRTMEM

Function

Reads the data from the virtual heap memory to a memory area pointed to by the DataPtr.

Declaration

VDskReadHeapData(var VDskPacket:VDskFile; DataPtr:pointer; VDskHeapPtr:longint)

Remarks

VDskPacket is the record type returned by VEMSOpenHeapFile or VDskOpenHeapFile. The DataPtr is of a pointer type that points to a memory buffer that will be overwritten by the transfer of data from the virtual heap. The VDskHeapPtr must be the virtual heap pointer created from VDskGetMem.

VDSKStatus will return a 0 if writing to the virtual heap was successful; otherwise, it will return a nonzero error code.

Restrictions

The Virtual Heap memory must be opened.

See also

VDSKGetMem, VDskFreeMem, VDskWriteHeapData

VDskCloseHeapFile procedure VIRTMEM

Function

Closes a virtual heap.

Declaration

VDskCloseHeapFile(var VDskPacket:VDskFile)

Remarks

VDskPacket is the record type returned by VEMSOpenHeapFile or VDskOpenHeapFile.

VDSKStatus will return a 0 if the virtual heap

operation was successful; otherwise, it will return a nonzero error code.

See also

VEMSOpenHeapFile, VDskOpenHeapFile

The Virtual Heap Error Function

The VDskError variable allows you to install a virtual heap error function, which gets called whenever the TEGL heap manager cannot complete an allocation request. VDskError is a pointer that points to a function with the following header:

```
    {$F+} {sh function} ReturnHeapError(code: longint) : word;  {$F-}
```

The virtual heap error function is installed by assigning its address to the VDskError variable:

```
    VDskError := ReturnHeapError;
```

The virtual heap error function gets called whenever any virtual function calls is unable to complete the request. The code parameter contains a code indicating which virtual heap function is in error. Check VDSKStatus to determine the severity of the error.

The standard virtual heap error function is set to return to the calling procedure.

If you are using the Virtual memory manager (next section), use the virtual memory error function rather then this error function to intercept virtual errors. The virtual memory manager relies on the standard q return to the calling procedure to check VDSKStatus to indicate whether to write to EMS or disk file.

The Virtual Memory Manager

The virtual memory manager is in constant use by TEGL windows to provide memory extensions for graphical images. Your program may use the virtual memory functions as an external heap, with the restriction that you do close the virtual memory file.

The following virtual memory functions will automatically select the storage medium when moving data to virtual memory. The data is moved to expanded memory if adequate space can be found, otherwise the data is

moved to one of the mass storage mediums. Both storage medium (EMS and Hard disk) are used if available.

UseHardDisk procedure VIRTMEM

Function

This function forces the virtual memory manager to use the hard disk as virtual memory, even if EMS is available.

Declaration

UseHardDisk(yesno:boolean)

Remarks

if the yesno is true, then the virtual memory manager will ignore the installed EMS, and only use the hard disk.

VDSKStatus will return a 0 if the virtual memory operation was successful; otherwise, it will return a nonzero error code.

MoveFromVirtual procedure VIRTMEM

Function

Moves a block of data from virtual back to normal memory.

Declaration

MoveFromVirtual(DataPtr:pointer; VirtualHeapPtr: longint)

Remarks

The DataPtr is any memory block allocated by GetMem or TEGLGetMem. VirtualHeapPtr is of the type longint, which is the address supplied by MovetoVirtual.

VDSKStatus will return a 0 if the virtual memory operation was successful; otherwise, it will return a nonzero error code.

See also

MoveToVirtual, FreeVirtual

MoveToVirtual function VIRTMEM

Function

Moves a block of data from memory to virtual memory.

Declaration

MoveToVirtual(DataPtr:pointer; HeapSize: longint)

Result type

MoveToVirtual returns a longint type, which is a physical address of the virtual block.

Remarks

The DataPtr is any memory block allocated by GetMem or TEGLGetMem. HeapSize is of the type longint, which is the size of the memory block that you are moving to virtual memory.

MoveToVirtual will automatically allocate EMS memory pages and open any virtual memory files (if needed) if this is the first time call to this procedure.

VDSKStatus will return a 0 if the virtual memory operation was successful; otherwise, it will return a nonzero error code.

See also

MoveFromVirtual, FreeVirtual

FreeVirtual procedure VIRTMEM

Function

Frees the virtual memory back to the virtual memory pool for reuse.

Declaration

FreeVirtual(VirtualHeapPtr:longint)

Remarks

VirtualHeapPtr is of the type longint, which is the address supplied by MovetoVirtual.

VDSKStatus will return a 0 if the virtual memory operation was successful; otherwise, it will return a

nonzero error code.

See also

MoveToVirtual, MoveFromVirtual

CloseVirtual procedure VIRTMEM

Function

Closes the virtual memory manager.

Declaration

CloseVirtual

Remarks

CloseVirtual shuts the operation of the virtual memory manager. The shut down procedure includes releasing allocated expanded memory pages and closing external virtual files.

VDSKStatus will return a 0 if the virtual memory operation was successful; otherwise, it will return a nonzero error code.

Restrictions

The procedure should not be called if the TEGLUnit is used.

TEGLMaxAvail Function VIRTMEM

Function

Returns the size of the largest block available in the upper heap.

Declaration

TEGLMaxAvail : LongInt;

VirtualMemUsed Function VIRTMEM

Function

Returns the amount of virtual memory allocated.

Declaration

```
VirtualMemUsed : LongInt;
```

Remarks

This is the total of virual memory allocated. On some systems this can be a combination of both EMS and Disk memory.

The Virtual Memory Error Function

The VirtualError variable allows you to install a virtual memory error function, which gets called whenever the virtual memory manager cannot complete a virtual function request. VirtualError is a pointer that points to a function with the following header:

```
    {$F+} {sh function} ReturnVirtualError(code: longint) : word;  {$F-}
```

The virtual memory error function is installed by assigning its address to the VirtualError variable:

```
    VDskError := ReturnVirtualError;
```

The virtual memory error function gets called whenever any virtual function calls is unable to complete the request. The code parameter contains a code indicating which virtual heap function is in error. Check VDSKStatus to determine the severity of the error.

The standard virtual memory error function is set to return to the calling procedure.

Resolving Fragments

The memory used by the heap is a dynamic and volatile part of your program. Memory is constantly allocated and de-allocated by the window manager along with allocation of dynamic variables, free space records, frame records, mouse click records, etc.

Although the virtual memory manager will provide almost unlimited windows, the concept is still limited by the number of window records that will fit in memory and whether the memory is contiguous or fragmented by allocated memory not under the control of the virtual memory manager.

Fragmentation occurs, when free memory blocks are separated by allocated blocks. Since certain allocated memory blocks cannot be moved or

de-allocated, fragmentation can cut down the largest block size available from the heap.

Without a proper control on memory fragmentation, an out of space error can still occur even when the virtual memory manager pages out all window images.

In order for the virtual memory Manager to provide large contiguous memory on the heap, two memory managers are used to partition the main heap memory. The normal Turbo Pascal heap manager is used to allocate simple memory blocks like frame information and virtual pointer information. The second, is the TEGL heap manager, used by the window manager to allocated large image buffers.

The function ReserveHugeMinimum partitions the heap memory into two parts by allocating a single byte between the minimum and upper memory. Normal allocations using Turbo Pascal Getmem will default to the lower areas by the methods that TP's uses to allocate memory. Turbo pascal will begin using the upper area when all lower memory area is used, thus it is not a restriction on TP's Getmem.
TEGLGetMem will only allocate memory from the upper areas.

ReserveHugeMinimum provides an elegant solution, that allows both memory managers to coexist.

ReserveHugeMinimum procedure VIRTMEM

Function

Partition the heap memory into lower and upper areas to reduce fragmentation.

Declaration

ReserveHugeMinimum(MinimumSize : longint)

Remarks

MinimumSize is of the type longint, which is the size calculated by adding (60 bytes for a window record) + the average mouse click and key clicks areas per window (20 bytes per each defined click) multiplied by the maximum number of window records opened at the same time + 4000 bytes (overhead for the virtual memory manager) plus any heap memory requirements by the application.

You are not expected to calculate the exact MinimumSize, but as a general rule of thumb, it seems that 12k is effective for most applications.

Chapter 13 - Virtual Memory Manager

Chapter 14 - Sizing and Sliders

Sizing and Sliding

The chapter has the procedures and functions that give the core for resizing frames and attachings sliders to them.

A slider is a moveable switch. They are quite often used to indicate up and down or left to right scrolls (as in a text editor). They can be attached to a window but are seperate, that is, they must be disposed of seperately.

Resizing frames adds a degree of complexity to maintaining frames in that the contents of the frame are lost when it is resized. Consequently, you need to code an event that specifically redraws a frame after resizing.

Resizable frames with slider bars require more work. It is up to the programmer to dispose of and then reattach new sliders to a frame after a resize. This presumably is all done within the event that redraws the window. This is not impossible, just careful thought is required when making these kinds of frames. The results will speak for themselves.

DefineResizeClickArea Procedure — TEGLSPEC

Function

Sets a mouse click area for resizing a frame.

Declaration

```
DefineResizeClickArea(ifs : ImageStkPtr;
  x,y,x1,y1 : Word; ResizeProc : CallProc);
```

Remarks

The ResizeProc must be defined. You cannot pass a NIL pointer. When a frame is resized its image is disposed and must be redrawn.

See also

DefineResizeMinMax.

Example

```
  DefineResizeClickArea(ifs,1,1,10,6,ReDrawEditor);
```

DefineResizeMinMax Procedure — TEGLSPEC

Function

Sets the minimum and maximum that a frame can be resized to.

Declaration

```
DefineResizeMinMax(ifs : ImageStkPtr; MinW,
  MinH,MaxW,MaxH : Word);
```

Remarks

MinW is the minimum width the frame is allowed if resized. MinH is the minimum height, MaxW is the maximum width, and MaxH is maximum height. Values are in pixels.

See also

DefineResizeClickArea.

Example

```
  DefineResizeMinMax(ifs,200,100,400,200);
```

DefineSliderArea Procedure TEGLSPEC

Function

Defines slider area.

Declaration

```
DefineSliderArea(ifs : ImageStkPtr; x,y,x1,y1,
  minx,miny,maxx,maxy: Word; SlideAction : CallProc);
```

Remarks

ifs is the frame the slider is attached to. x,y, x1, y1 is the slider click area. minx, miny, maxx, maxy are the bounds the slider can be moved in. Coordinates are frame relative. SlideAction is the event that is called when the slider is moved.

The MsClickPtr that is passed to SlideAction contains the new slider position. These coordinates can be used to determine the correct action to taken.

Restrictions

This procedure only sets the area for the slider and its bounds. It is up to the programmer to draw the slider bar and the slider. The slider bar must be drawn before the call to DefineSliderArea. Then after draw this the

slider. The toolkit will look after moving the slider once it has been drawn.

See also

SetSlidePostion

Example

DropSliders Procedure TEGLSPEC

Function

Removes all sliders from a frame.

Declaration

DropSliders(ifs : ImageStkPtr);

Remarks

DropSliders should be called before you drop a frame or resize it.

Restrictions

See also

Example

```
  DropSliders(ifs);
```

FindSliderFS Function TEGLSPEC

Function

Finds a slider on a frame.

Declaration

FindSliderFS(ifs : ImageStackPtr; ms : MsClickPtr):
  SliderPtr;

Remarks

Returns the SliderPtr associated with ms on the frame. This can be used from within an event that is called when a slider is moved. With the SliderPtr

you can determine the relative position of the slider without having to examine any other variables.

Restrictions

See also

Example

ResizeFrame Procedure TEGLSPEC

Function

Allocates a new buffer for a frame.

Declaration

ResizeFrame(ifs : ImageStkPtr; x,y,x1,y1 : Word);

Remarks

x, y, x1, y1 are the new coordinates of the frame.

Restrictions

The frame image is hidden then disposed.

See also

DefineResizeMinMax

Example

SelectAndMoveFrame Event TEGLSPEC

Function

An event that allows the frame to be moved.

Declaration

```
SelectAndMoveFrame(ifs: ImageStkPtr; ms: MsClickPtr):
  Word;
```

Remarks

Note that this is an event. You would not directly call it but rather would pass it with a DefineMouseClickArea.

See also

DefineMouseClickArea.

Example

```
{ -- the top 10 pixels across the frame ifs is set to SelectAndMoveFrame }

DefineMouseClickArea(ifs,0,0,ifs^.x1,10,TRUE,SelectAndMoveFrame,MSClick);
```

SetSlidePosition Procedure TEGLSPEC

Function

Moves a slider to a new position.

Declaration

SetSlidePosition(ifs : ImageStkPointer; x,y : Word);

Remarks

x,y are relative coordinates within the frame and must be within the slider bar.

See also

DefineSliderArea.

Miscellaneous Functions

CheckCtrlBreak Procedure TEGLUNIT

Function

Checks task handler.

Declaration

CheckCtrlBreak;

Remarks

Normally this routine does not have to be called, but if you have section of code that is going through a long loop you should insert it there.

If your program has events that are activated after a certain number of timer ticks have passed then a call to CheckCtrlBreak will allow their processing.

The TEGL Windows Toolkit does not process timer interrupt tasks directly, rather a flag is set and the task is performed when it is safe (ie. no frames are being updated and no memory swaps are begin processed).

Example

```
VAR x : LongInt;

  FOR x := 1 TO 20000000 DO
    BEGIN
      { -- do your stuff }
      CheckCtrlBreak;   { --  allow processing of other tasks }
    END;
```

CheckCtrlBreakFS Procedure TEGLUNIT

Function

Sets an event to call when Ctrl-Break is pressed.

Declaration

CheckCtrlBreakFS(p : CallProc);

Remarks

P is an event and works like any other. You can determine within it what processing should take place (Halt, Continue, Close files, etc..).

Example

See InitTEGL in TEGLEasy.

DropTimerTick Procedure TEGLUNIT

Function

Removes an event set with SetTimerTick.

Declaration

DropTimerTick(Ticks : Word; P : CallProc);

Remarks

Both Ticks and P must be identical to the orginal call for the event to be removed.

See also

SetTimerTick.

Example

```
  DropTimerTick(18,BackGroundClock);
```

NilUnitProc Event TEGLUNIT

Function

A place holder for events that have not been coded.

Declaration

NilUnitProc;

Remarks

NilUnitProc can be used wherever an event handler

is called for. This can be a place holder or it can be where event is desired but a parameter is required.

Example

```
{ -- a line in a menu that is never selected or active }
DefineOptions(filem,'--',false,nilunitproc);
```

OverLapArea Function — TEGLUNIT

Function

Returns the area that is occupied by two sets of coordinates.

Declaration

```
OverlayArea(ax, ay, ax1, ay1, bx, by, bx1,b y1 : Word;
  VAR cx, cy, cx1, cy1 : Word) : Boolean
```

Remarks

a and b coodinates are the areas to test.
If they overlap then the area is return in the c coordinates and the function returns true, otherwise the function returns false and the c coordinates are undetermined.

This is an advanced function that normally would not be used.

SetTimerTick Procedure — TEGLUNIT

Function

Sets an event to be called periodically.

Declaration

```
SetTimerTick(Ticks : Word; p : CallProc;
  ifs : ImageStkPtr; ms : MsClickPtr);
```

Remarks

Ticks is how many timer ticks to wait before

begin called. p is the event to call. ifs and ms are passed to p.

See also

DropTimerTick.

Example

```
SetTimerTick(18,BackGroundClock,NIL,NIL);
```

TGraph

The TGraph unit provides a subset of the functions in the Graph unit provided with Turbo Pascal.

TGraph does not have to be used if you are using Turbo Pascal. If your program requires elaborate graphics drawing and painting then the Graph unit is needed. If, however, your graphics need are simpler then TGraph may provide all that is needed. If this is the case your program can be as much as 25K smaller by using TGraph exculsively. See the appendix Condtional Compilation for directions on building the toolkit without using the Graph unit.

If you are programming with Microsoft's Quick Pascal then TGraph is necessary. Depending on the defines in the file switches.inc (see the appendix Conditional Compilation) TGraph acts as stand-alone or maps graphics calls to the equivalent MSGraph routine.

Bar Procedure — TGRAPH

Function

Draws a bar using the current fill style and color.

Declaration

Bar(x1, y1, x2, y2: Integer);

Remarks

Draws a filled in bar using the pattern and color defined by SetFillStyle or SetFillPattern.

See also

SetFillStyle, SetFillPattern

CloseGraph Procedure — TGRAPH

Function

Shuts down the graphics system.

Declaration

CloseGraph

Remarks

The screen mode is restored to the original mode before

graphics were initialized.

DetectGraph Procedure — TGRAPH

Function
Detects graphics hardware.

Declaration
```
DetectGraph(VAR GraphDriver, GraphMode : Integer);
```

Remarks
Returns the detected driver and mode value that can be passed to InitGraph which will change to graphics mode. If no graphics hardware is found or the graphics hardware is not supported then a call to GraphResult will return a value of -2 (grNotDetected).

See also
InitGraph, GraphResult

GetBkColor Function — TGRAPH

Function
Returns the current background color.

Declaration
```
GetBkColor : word;
```

Remarks
Background colors can range from 0 to 15.

See also
GetColor, SetBkColor, SetColor

GetColor Function — TGRAPH

Function

Returns the color value passed to the previous call to SetColor.

Declaration

GetColor : Word;

Remarks

Drawing colors can range from 0 to 15.

See also

SetColor

GetFillPattern Procedure TGRAPH

Function

Returns the last fill pattern set by the last call to SetFillPattern.

Declaration

GetFillPattern(VAR FillPattern : FillPatternType);

Remarks

FillPatternType is

```
TYPE
  FillPatternType = array[0..8] of byte;
```

See also

SetFillPattern, GetFillSettings

GetGraphMode Function TGRAPH

Function

Returns the current graphics mode.

Declaration

GetGraphMode : Integer;

Remarks

Returns the current graphics mode set by InitGraph or SetGraphMode.

See also

TGraph

DetectGraph, InitGraph, RestoreCrtMode,
SetGraphMode

GetMaxX Function — TGRAPH

Function

Returns the pixel width (minus 1) of the current graphics driver and mode.

Declaration

GetMaxX : Integer;

Remarks

GetMaxX can be used to determine the boundaries of the screen.

See also

GetMaxY, GetX, GetY

GetMaxY Function — TGRAPH

Function

Returns the pixel height (minus 1) of the current graphics driver and mode.

Declaration

GetMaxY : Integer;

Remarks

GetMaxY can be used to determine the boundaries of the screen.

See also

GetMaxX, GetX, GetY

GetTextSettings Procedure — TGRAPH

Function

Returns the current text settings.

Declaration

GetTextSettings(VAR TextInfo : TextSettingsType);

Remarks

TextSettingsType contains fields for the font, direction, size and justification that was set by SetTextStyle and SetTextJustify.

See also

SetTextJustify, SetTextStyle

ImageSize Function \ TGraph

Function

Returns the number of bytes required to store a rectangular region of the screen.

Declaration

ImageSize(x1, y1, x2, y2: Integer): Word;

Remarks

x1,y1,x2,y1 defines the area on the screen.

GraphResult Function TGRAPH

Function

Returns the error code for the last graphics operation.

Declaration

GraphResult : Integer;

Remarks

GraphResult is reset to zero after it has been called. The user may want to store it into a temporary variable before testing it.

InitGraph Procedure TGRAPH

Function

Initializes the graphics system and sets the hardware to

graphics mode.

Declaration

```
InitGraph(VAR GraphDriver : Integer;
  VAR GraphMode: Integer; DriverPath : String);
```

Remarks

If GraphDriver is equal to 0 (Detect) then a call is made to DetectGraph. If supported hardware is detected then the graphics system is initialized and a graphics mode is selected.

The parameter DriverPath is provided for compatibility with Graph, it is not used, all drivers are linked in.

See also

DetectGraph, CloseGraph

Line Procedure TGRAPH

Function

Draws a line from x1, y1 to x2, y2.

Declaration

```
Line(x1, y1, x2, y2 : Integer);
```

Remarks

Draws a line in the color set by SetColor

OutTextXY Procedure TGRAPH

Function

Sends a string to the screen.

Declaration

```
OutTextXY(x,y : Integer; TextString: String);
```

Remarks

TextString is output at the screen location x,y.

OutTextXY uses the options set by SetTextJustify.

See also

SetTextJustify, GetTextSettings

Rectangle Procedure TGRAPH

Function

Draws a rectangle using the current color.

Declaration

Rectangle(x1, y1, x2, y2 : Integer);

Remarks

x1,y1 define the upper left corner of the rectangle, and x2,y2 define the lower right corner.

See also

SetColor

RestoreCrtMode Procedure \ TGraph

Function

Restore the screen mode.

Declaration

RestoreCrtMode;

Remarks

Restore the screen mode to its original state before graphics was initialized.

See also

DetectGraph, InitGraph

SetBkColor Procedure TGRAPH

Function

Sets the backgound color.

Declaration

Remarks

Background colors may range from 0 to 15.

See also

GetBkColor,SetColor

SetColor Procedure TGRAPH

Function
Set the drawing color.

Declaration
SetColor(Color : Word);

Remarks
Drawing colors may range from 0 to 15.

See also
GetColor

SetFillPattern Procedure TGRAPH

Function
Selects a user-defined fill pattern.

Declaration
SetFillPattern(Pattern : fillPatternType; Color: Word);

Remarks
Sets the pattern and color for all filling done by Bar.

See also
GetFillPattern, SetFillStyle

SetFillStyle Procedure TGRAPH

Function
Sets the fill pattern and color.

Declaration
SetFillStyle(Pattern : Word; Color: Word);

Remarks
Set the pattern and color for all filling done by Bar. There are 12 fill patterns available.

See also
GetFillSettings

SetTextJustify Procedure TGRAPH

Function

Sets text justification values used by OutTextXY.

Declaration

SetTextJustify(Horiz, Vert: Word);

Remarks

The default justification settings are SetTextJustify(
LeftText, TopText).

See also

GetTextSettings, OutTextXY

APPENDICES

Appendix A - Overlapping Graphics

There are many methods in creating and managing overlapping windows, however the end result to the user must be in the context of windows that form independent layers on a single display.

This section discusses the method that is used with the TEGL Windowing Manager.

Video Buffers

The video buffer is a block of memory where displayable data is stored. A program may read and write to the video buffer in the same way it accesses any other memory.

The video display circuitry updates the screen by continually reading the data in the video buffer and translating the bit information to the screen. Each group of bits in the video buffer specifies the color and brightness of a particular location on the screen. A particular location on the screen is known as a pixel. If a program changes the contents of the video buffer, the screen reflects the change immediately.

Because you have control over each pixel in the displayed image, you can construct complex geometric images, fill arbitrary areas of the screen with blends of colors, or create animated images that moves across the screen.

We may think of windows as multiple video buffers, the distinction is that, with the TEGL Windows Toolkit, only 1 video buffer is used. To create a window effect, we must physically copy and move display data to and from a single video buffer, overlaying the images as we would layout images on paper.

Windows

Windows are simply predefined rectangular areas of the screen. A window manager is a coordinator that ensures that images related to a window are saved (stored in memory) before other overlapping images writes to the screen. When a window is closed, the underlying image is copied back to screen video buffer.

The basis of a window manager is the copying and restoring of multiple areas of the screen.

Frames

An EGA video has a maximum resolution of 640 pixels horizontal by 350

pixels. The coordinates are specified as (x,y) and (x1,y1), where x and y are the horizontal and vertical position respectively. The position is relative to upper left coordinate which has a coordinate value of (0,0).

```
(x,y)
  +---------+
  |(y)      |
  ||        |
  ||        |
  |*        |
  |         |
  |(x)----->|
  +---------+
         (x1,y1)
```

A Frame Stack

A frame stack is a list with each entry representing a screen area. Each entry contains information and data that is required by the window manager to coordinate the overlaps between frames.

The order of the list is in the same order as the frames are stacked on the screen.

A Simple Window Manager

This section talks about creating a simple window manager. We will use the following example to see how we can update frame (A) independent of the other 3 frames.

The following frames have called PUSHIMAGE to save the underlying graphics.

```
          +-----------------+
+---------+--+              |
|            |              |
|            |--+           |
|            |B |  A        |
| D          |--+           |
|            |              |
|            |--+           |
|            |C |           |
|            |--+           |
|            |              |
+---------+--+              |
          |                 |
          +-----------------+
```

In order for Frame (A) to be updated, the image for Frame (D) is saved, and Frame (D) is erased from the screen
by restoring the the underlying image that was saved previously.

```
            +-----------------+
            |                 |
            |                 |
      +-----|-----+           |
      |  B  |     |   A       |
      +-----|-----+           |
            |                 |
      +-----|-----+           |
      |  C  |     |           |
      +-----|-----+           |
            |                 |
            |                 |
            |                 |
            +-----------------+
```

The image for Frame (C) is saved, and Frame (C) is erased from the screen by restoring the the underlying image that was saved previously.

```
            +-----------------+
            |                 |
            |                 |
      +-----|-----+           |
      |  B  |     |   A       |
      +-----|-----+           |
            |                 |
            |                 |
            |                 |
            |                 |
            |                 |
            |                 |
            |                 |
            +-----------------+
```

The image for Frame (B) is saved, and Frame (B) is erased from the screen by restoring the the underlying image that was saved previously.

```
            +-----------------+
            |                 |
            |                 |
            |                 |
            |         A       |
            |                 |
            |                 |
            |                 |
            |                 |
            |                 |
            |                 |
            |                 |
            |                 |
            +-----------------+
```

The composite image of (A) is now complete and can be updated. The images (B), (C) and (D) are restored by reversing the above steps.

In the earlier generations of TEGL, this formed the basis of the stacked frame concept (the removal of images that overlaps the current).

Partial Image Update

As you can imagine, this process is slow and causes a lot a of unnecessary updates to the screen. With the foundation of q a simple window manager, we can now begin to refine this process.

Partial image update is removing only the intersection portion of the frames from the screen by extracting a section of the saved image.

The following shows the intersection of D,C and B that is needed to be replaced on the screen.

```
+--+
|D |
|  |  +------+
|  |  |B     |
|  |  +------+
|  |  +------+
|  |  |C     |
|  |  +------+
+--+
```

Partial Image (D) is replaced first, followed by Partial (C) and (B).

Refined Partial Image Update

Since we are only interested in the composite image of (A), there is still a lot of unnecessary update to the screen.

Imagine a notepad and you wish to write on the fifth page of the notepad. The fastest way to lift up five pages in a group, write, and close the notepad.

So let's split image (D) into 5 pieces.

```
+--+
|D1|
+--+
+--+--+
|D2| B|
+--+--+
+--+
|D3|
+--+
+--+--+
```

```
      |D4| C|
      +--+--+
      +--+
      |D5|
      +--+
```

Notice the double pages of (D2)(B) and (D4)(C). Now we only need to replace (D1), (B), (D3), (C) and (D5). We don't need to replace (D2) and (D4) because (B) and (C) has already restored the composite image of (A).

TEGL was further refined to (cut out) only the pieces that needs to be replaced, thus removing all unnecessary updates to the screen.

A Refined Partial Image Update Algorithm

check for condition where by replacing the bottom image will replace the top image. eg.

```
                             +----------------+
                             |1 +---------+    |
                             |  | 2 +-----+    |
                             |  |   |  3  |    |
                             |  +---+-----+    |
                             +----------------+
```

Replacing 3 will be redundant, since we want to update 1, replacing 2 will remove both 2 and 3.

check if we can begin trim the ends off one of the overlapped images to reduce the size that we need to replace.

```
+---------+ +---------+ +---------+  +---------+ +---+---------+ +---------+---+
|   +-----+ | +-----+ | +-----+   |  |     :   | |   | :       | |       : |   |
+---|.....| +-|.....|-+ |.....|---+  +---------+ |   +---------+ +---------+   |
    |     |   |     |   |     |      |     |     |     |                 |     |
    |     |   |     |   |     |      +-----+     +-----+                 +-----+
    +-----+   +-----+   +-----+      +-----+     +-----+                 +-----+
+---+-----+ +-+-----+-+ +-----+---+  +---------+ |   +---------+ +---------+   |
|   |     | | |     | | |     |   |  |     :   | |   | :       | |       : |   |
+---|.....| +-|.....|-+ |.....|---+  +---------+ |   +---------+ +---------+   |
    |     |   |     |   |     |      +-----+     +-----+                 +-----+
    +-----+   +-----+   +-----+      +-----+              +-----+     +-----+
    +-----+   +-----+   +-----+      |     |              |     |     |     |
    |     |   |     |   |     |      +---------+  +---------+   | +---------+
+---|.....| +-|.....|-+ |.....|---+  |     :   |  |       : |   | |   :     |
|   |     | | |     | | |     |   |  +---------+  +---------+---+ +---------+
+---+-----+ +-+-----+-+ +-----+---+  +-----+            +-----+  +---------+
    +-----+   +-----+   +-----+      |     |        +---------+  |   :     |
    |     |   |     |   |     |      |   +---------+|   :     |  +---------+
    |     |   |     |   |     |      |   | :       |+---------+      |     |
```

```
+---|.....| +-|.....|-+ |.....|---+  +---+---------+    +-----+      +-----+
|   +-----+ | +-----+ | +-----+   |
+---------+ +---------+ +---------+
```

create an new insert that has one end trimmed and repeat steps 1 through 3 to cut the images into the necessary pieces.

```
    +-----+   +-----+   +-----+     +---------+   +-----+     +-----+
+---|     | +-|     |-+ |     |---+ |       : | +---------+   |     |
|   |     | | |     | | |     |   | +---------+ |       : | +---------+
+---|.....| +-|.....|-+ |.....|---+   |     |   +---------+ |       : |
    +-----+   +-----+   +-----+       +-----+     +-----+   +---------+
                                +-----+-------+                 +-------+-----+
        +-----+ +-----+         |     |       | +-----+ +-----+ |       |     |
+-------|.....| |.....|-------+ |.....|-------+ |     | |     | +-------|.....|
|       |     | |     |       | |     | +-------|.....| |.....|-------+ |     |
+-------|     | |     |-------+ +-----+ |       |     | |     |       | +-----+
        +-----+ +-----+                 +-------+-----+ +-----+-------+
+---------++-----++---------+    +---------+    +---------++-----++---------+
|.........||     ||.........|---+|.........|+---|.........||     ||.........|
+---------+|     |+---------+   |+---------+|   +---------+|     |+---------+
    |     ||.....|-------+|     |  |     |  |     ||+-------|.....||     |
    |     |+-----+       ||     |  |     |  |     ||       +-----+|     |
    +-----+    +---------++-----+  +-----+  +-----++---------+    +-----+
+-----+    +-----++---------+      +-----+      +---------++-----+    +-----+
|     |    |     ||       +-----+  |     |  +-----+       ||     |    |     |
|     |    |     |+-------|.....|  |     |  |.....|-------+|     |    |     |
+---------+|   +---------+|     |+---------+|     |+---------+   |+---------+
|.........|+---|.........||     ||.........||     ||.........|---+|.........|
+---------+    +---------++-----++---------++-----++---------+    +---------+
```

The only time that we are unable to split an overlapping image is when the image overlaps by 1 pixel.

```
                                    +---------+
                                    |         |
                         +----------|         |-----------+
                         |          +----+---------+      |
                         +---------------|         |------+
                                         |         |
                                         +---------+
```

A Quick Run through the algorithm

The procedure to handle the splitting of images is called StackOverlaps. StackOverlaps works in the following fashion:

```
                   Top (Stackptr)*                              Bottom
               +--+---------+-----+---+        Top      +----------------+
               |D |x,y,x1,y1|image|...|       +---------+--+             |
```

APPENDICES

```
               +--+---------+-----+---+       |            |             |
               +--+---------+-----+---+       |            |--+          |
               |C |x,y,x1,y1|image|...|       |            |B |  A       |
               +--+---------+-----+---+       | D          |--+          |
               +--+---------+-----+---+       |            |             |
               |A |x,y,x1,y1|image|...|       |            |--+          |
               +--+---------+-----+---+       |            |C |          |
               +--+---------+-----+---+       |            |--+          |
               |A |x,y,x1,y1|image|...|       |            |             |
               +--+---------+-----+---+       +---------+--+             |
                   Bottom                               |                |
                                                        +----------------+
```

PrepareForUpdate(A) creates temporary stack entries:

```
                   Top (Stackptr)*
               +--+---------+-----+---+
        +----> |B1|x,y,x1,y1|image|...|
        |      +--+---------+-----+---+
        |      +--+---------+-----+---+                   +----------------+
        | +--> |C1|x,y,x1,y1|image|...|        +---------+--+             |
        | |    +--+---------+-----+---+        |         |D1|             |
        | |    +--+---------+-----+---+        |         +-----+          |
        | | +> |D1|x,y,x1,y1|image|...|        |         |B1   |  A       |
        | | |  +--+---------+-----+---+        | D       +-----+          |
        | | |  +--+---------+-----+---+        |         +-----+          |
        | | +> |D |x,y,x1,y1|image|...|        |         |C1   |          |
        | |    +--+---------+-----+---+        |         +-----+          |
        | |    +--+---------+-----+---+        |         |  |             |
        | +--> |C |x,y,x1,y1|image|...|        +---------+--+             |
        |      +--+---------+-----+---+                  |                |
        |      +--+---------+-----+---+                  +----------------+
        +----> |B |x,y,x1,y1|image|...|
               +--+---------+-----+---+
               +--+---------+-----+---+
               |A |x,y,x1,y1|image|...|
               +--+---------+-----+---+
                   Bottom
```

Begin Cutting and Eliminating: Comparing only the overlapped images.

```
               +--+---------+-----+---+             +--+        +--+
        +----> |B1|x,y,x1,y1|image|...| Bottom      |D1|        |D1|
        |      +--+---------+-----+---+     Image   |  |--+     |  |
        |      +--+---------+-----+---+             |  |B1|     |  |
        | +--> |C1|x,y,x1,y1|image|...|             |  |--+     |  |
        | |    +--+---------+-----+---+             |  |        |  |
        | |    +--+---------+-----+---+             |  |        |  |--+
        | | +> |D1|x,y,x1,y1|image|...| Top Image   |  |        |  |C1|
        | | |  +--+---------+-----+---+             |  |        |  |--+
                                                    |  |        |  |
                                                    +--+        +--+
```

StackOverlaps compares B1 with D1, B1 with C1 and
C1 with D1 for overlaps.

eliminate redundant overlaps

Appendix B - Heap Management

One of the major problems with window management is the amount of dynamic memory that is allocated and de-allocated. Memory is constantly fragmented with records, dynamic variables, and window images, thus reducing the largest block size over a period of time.

```
           Empty Heap Memory                     Fragmented Heap Memory

           Top of DOS Memory                       Top of DOS Memory
FreePtr-> +-----------------+                     +-----------------+
          |        *        |                     |-----------------| 1--+
          |                 |           FreePtr-> |-----------------| 2--| +
          |                 |                     |                 |    | |
          |                 |                     |    Free Space   |    | |
          |                 |                     |       60k       |    | |
          |                 |           HeapPtr-> |-----------------|    | |
          |        *        |                     |-----------------|    | |
          |     MaxAvail    |                     |                 |    | |
          |      341k       |                     |    Free Space   |    | |
          |        *        |                     |       70k       |    | |
          |                 |                     |                 |    | |
          |                 |                     |-----------------| 2<-| +
          |                 |                     |-----------------|    |
          |                 |                     |    Free Space   |    |
          |                 |                     |        *        |    |
          |                 |                     |     MaxAvail    |    |
          |                 |                     |       102k      |    |
          |                 |                     |        *        |    |
          |        *        |                     |                 |    |
HeapPtr-> |-----------------| <-----HeapOrg-----> |-----------------| 1<-+
          |   * Program *   |                     |   * Program *   |
```

This chapter will discuss how the normal Turbo Pascal heap manager and the TEGL heap manager can coexist, and how ReserveHugeMinimum reduces the fragmentation that occurs.

Turbo Pascal Heap Manager

There are only two main pointers that manages the heap. The HeapPtr points to the end of the last memory block. FreePtr points to a list of free memory blocks that can be re-used.

```
           Top of DOS Memory
          +-----------------+
          |-----------------|1--+
FreePtr-> |-----------------|2--|+
          |                 |   ||
          |    Free Space   |   ||
          |       60k       |   ||
HeapPtr-> |-----------------|   ||
          |-----------------|   ||
          |                 |   ||
          |    Free Space   |   ||
          |       70k       |   ||
          |                 |   ||
          |-----------------|2<-|+
          |-----------------|   |
          |    Free Space   |   |
          |        *        |   |
          |     MaxAvail    |   |
          |      102k       |   |
          |        *        |   |
          |                 |   |
          |-----------------|1<-+
          |   * Program *   |
```

When memory is requested from the Turbo Pascal Heap Manager, a sequential scan of the Freeptr chain is made to see if any of the free memory space can be re-used. Any free space that satisfy the requested size will be used.

The free space is then reduced by the allocation size and removed from the FreePtr chain if the block is completely allocated.

GETMEM(102k)

```
           Top of DOS Memory                       Top of DOS Memory
          +-----------------+                     +-----------------+
          |-----------------|1--+      FreePtr->  |-----------------|2--+
FreePtr-> |-----------------|2--|+                |                 |   |
          |                 |   ||                |                 |   |
          |    Free Space   |   ||                |    Free Space   |   |
          |       60k       |   ||                |       60k       |   |
HeapPtr-> |-----------------|   ||     HeapPtr->  |-----------------|   |
          |-----------------|   ||                |-----------------|   |
          |                 |   ||                |                 |   |
          |    Free Space   |   ||                |    Free Space   |   |
          |       70k       |   ||                |       70k       |   |
          |                 |   ||                |                 |   |
```

```
         |-----------------|2<-|+              |-----------------|2<-+
         |-----------------|   |               |/////////////////|
         |    Free Space   |   |               |/////////////////|
         |        *        |   |               |/////////////////|
         |     MaxAvail    |   |               |/////////////////|
         |      102k       |   |               |/////////////////|
         |        *        |   |               |/////////////////|
         |                 |   |               |/////////////////|
         |-----------------|1<-+               |-----------------|
         |   * Program *   |                   |   * Program *   |
```

GETMEM(20k)

```
           Top of DOS Memory                     Top of DOS Memory
         +-----------------+                   +-----------------+
         |-----------------|1--+               |-----------------|1--+
FreePtr-> |-----------------|2--|+    FreePtr-> |-----------------|2--|+
         |                 |   ||              |                 |   ||
         |    Free Space   |   ||              |    Free Space   |   ||
         |       60k       |   ||              |       60k       |   ||
HeapPtr-> |-----------------|   ||    HeapPtr-> |-----------------|   ||
         |-----------------|   ||              |-----------------|   ||
         |                 |   ||              |    Free Space   |   ||
         |    Free Space   |   ||              |       50k       |   ||
         |       70k       |   ||              |-----------------|2<-|+
         |                 |   ||              |/////////////////|   |
         |-----------------|2<-|+              |-----------------|   |
         |-----------------|   |               |-----------------|   |
         |    Free Space   |   |               |    Free Space   |   |
         |        *        |   |               |        *        |   |
         |     MaxAvail    |   |               |     MaxAvail    |   |
         |      102k       |   |               |      102k       |   |
         |        *        |   |               |        *        |   |
         |                 |   |               |                 |   |
         |-----------------|1<-+               |-----------------|1<-+
         |   * Program *   |                   |   * Program *   |
```

When memory is released (freed), the Turbo Pascal Heap Manager sequentially scans the Freeptr chain to see if any of the free memory space is adjacent to the memory block that is being freed.

FREEMEM(A)

```
           Top of DOS Memory                     Top of DOS Memory
         +-----------------+                   +-----------------+
         |-----------------|1--+               |-----------------|1--+
         |-----------------|2--|+              |--------x--------|2--|+
         |--------x--------|3--||+             |-----------------|5--||--+
         |-----------------|4--|||+  FreePtr-> |-----------------|4--||+ |
FreePtr-> |-----------------|5--||||+           |        *        |   ||| |
         |        *        |   |||||           |                 |   ||| |
         |                 |   |||||           |                 |   ||| |
```

```
          |                 |   |||||
          |        *        |   |||||
HeapPtr-> |-----------------|   |||||
          |+++++++++++++++++|   |||||
          |+++++++++++++++++|2<-|+|||
    (A)>>>|+++++++++++++++++|   | |||
          |+++++++++++++++++|3<-|-+||
          |+++++++++++++++++|   |  ||
          |+++++++++++++++++|5<-|--|+
          |+++++++++++++++++|   |  |
          |+++++++++++++++++|1<-+  |
          |+++++++++++++++++|      |
          |+++++++++++++++++|4<----+
          |   * Program *   |
```

```
          |                 |   |||  |
          |        *        |   |||  |
HeapPtr-> |-----------------|   |||  |
          |+++++++++++++++++|   |||  |
          |+++++++++++++++++|2<-|+|  |
          |                 |   | |  |
    (A)>>>|-----------------|   | |  |
          |+++++++++++++++++|   | |  |
          |+++++++++++++++++|5<-|-|-+
          |+++++++++++++++++|   | |
          |+++++++++++++++++|1<-+ |
          |+++++++++++++++++|     |
          |+++++++++++++++++|4<---+
          |   * Program *   |
```

If adjacent memory is found, the free space pointer is removed from the FreePtr chain and Turbo Pascal's heap manager takes the most recent entry and moves it to fill the now empty position. The size and the original pointer (A) is adjusted to reflect a new pointer position and size.

```
           Top of DOS Memory
          +-----------------+
          |-----------------|1--+
          |-----------------|4--|-+
FreePtr-> |-----------------|5--|-|+
          |        *        |   | ||
          |                 |   | ||
          |                 |   | ||
          |                 |   | ||
          |                 |   | ||
          |                 |   | ||
          |        *        |   | ||
HeapPtr-> |-----------------|   | ||
          |-----------------|   | ||
          |                 |   | ||
          |                 |   | ||
    (A)>>>|-----------------|   | ||
          |+++++++++++++++++|   | ||
          |+++++++++++++++++|5<-|-|+
          |+++++++++++++++++|   | |
          |+++++++++++++++++|1<-+ |
          |+++++++++++++++++|     |
          |+++++++++++++++++|4<---+
          |   * Program *   |
```

```
           Top of DOS Memory
          +-----------------+
          |-----------------|1--+
          |-----------------|4--|-+
          |-----------------|5--|-|+
FreePtr-> |-----------------|A--|+||
          |                 |   ||||
          |                 |   ||||
          |                 |   ||||
          |                 |   ||||
          |                 |   ||||
          |        *        |   ||||
HeapPtr-> |-----------------|   ||||
          |+++++++++++++++++|   ||||
          ||||||||||||||||||    ||||
          ||||||||||||||||||    ||||
          |+++++++++++++++++|A<-|+||
          |+++++++++++++++++|   | ||
          |+++++++++++++++++|5<-|-|+
          |+++++++++++++++++|   | |
          |+++++++++++++++++|1<-+ |
          |+++++++++++++++++|     |
          |+++++++++++++++++|4<---+
          |   * Program *   |
```

When all possible adjacent blocks have been removed, the Turbo Pascal heap manager checks if the end of memory block is equal to the HeapPtr. If not, a free space entry is added to the bottom of FreePtr.

FREEMEM(B)

APPENDICES

```
           Top of DOS Memory                      Top of DOS Memory
          +-----------------+                    +-----------------+
          |-----------------|1--+                |-----------------|1--+
          |-----------------|4--|-+              |-----------------|4--|-+
          |-----------------|5--|-|+   FreePtr-> |-----------------|5--|-|+
FreePtr-> |--------x--------|A--|+||             |        *        |   | ||
          |        *        |   ||||             |                 |   | ||
          |                 |   ||||             |                 |   | ||
          |                 |   ||||             |                 |   | ||
          |                 |   ||||             |                 |   | ||
          |                 |   ||||             |                 |   | ||
          |        *        |   ||||             |                 |   | ||
HeapPtr-> |-----------------|   ||||             |                 |   | ||
    (B)>>>|-----------------|   ||||             |                 |   | ||
          |                 |   ||||             |                 |   | ||
          |                 |   ||||             |        *        |   | ||
          |-----------------|A<-|+||   HeapPtr-> |-----------------|   | ||
          |+++++++++++++++++|   | ||             |+++++++++++++++++|   | ||
          |+++++++++++++++++|5<-|-|+             |+++++++++++++++++|5<-|-|+
          |+++++++++++++++++|   | |              |+++++++++++++++++|   | |
          |+++++++++++++++++|1<-+ |              |+++++++++++++++++|1<-+ |
          |+++++++++++++++++|     |              |+++++++++++++++++|     |
          |+++++++++++++++++|4<---+              |+++++++++++++++++|4<---+
          |   * Program *   |                    |   * Program *   |
```

As you may note that the free space pointer chain is almost on a first in first out basis. The most recent freed block is used first. Thus heap activities is dependant on localize freeing of memory. A more effecient method is sorting the free space entries, so that attempts to allocate space will always be towards the lower part of the heap memory. However this is not the most effective method. If a single non-movable record is allocated in the middle of the heap, this will fragment the heap into two parts.

TEGL Heap Manager

The TEGL Heap Manager is slightly different in its management methods. Allocation of memory is always attempted between HeapPtr and FreePtr before searching for free space within the FreePtr chain.

GETMEM(20k)

```
           Before                    Getmem                    TEGLGetMem
          +-------+                 +-------+                 +-------+
          |-------|1--+             |-------|1--+             |-------|1--+
FreePtr-> |-------|2--|+  FreePtr-> |-------|2--|+  FreePtr-> |-------|2--|+
          |   *   |   ||            |   *   |   ||            |  40k  |   ||
          |  60k  |   ||            |  60k  |   ||            |-------|   ||
          |   *   |   ||            |   *   |   ||            |///////|   ||
HeapPtr-> |-------|   ||  HeapPtr-> |-------|   ||  HeapPtr-> |-------|   ||
```

```
          |-------|   ||            |-------|   ||            |-------|   ||
          |       |   ||            |   *   |   ||            |       |   ||
          |   *   |   ||            |  50k  |   ||            |   *   |   ||
          |  70k  |   ||            |-------|2<-|+            |  70k  |   ||
          |   *   |   ||            |///////|   |             |   *   |   ||
          |-------|2<-|+            |-------|   |             |-------|2<-|+
          |-------|   |             |-------|   |             |-------|   |
          |       |   |             |       |   |             |       |   |
          |       |   |             |       |   |             |       |   |
          |   *   |   |             |   *   |   |             |   *   |   |
          | 102k  |   |             | 102k  |   |             | 102k  |   |
          |   *   |   |             |   *   |   |             |   *   |   |
          |       |   |             |       |   |             |       |   |
          |-------|1<-+             |-------|1<-+             |-------|1<-+
          |Program|                 |Program|                 |Program|
```

When memory is released (freed), the TEGL Pascal Heap Manager is similar to Turbo Pascal Heap Manager in that adjacent memory block are combined by scanning through the Freeptr chain. However the difference that is noticeble immediately, is the sorted order of the free space pointers in comparison to the FIFO structure of TP's.

FREEMEM(A)

```
           Top of DOS Memory                       Top of DOS Memory
          +-----------------+                     +-----------------+
          |-----------------|1--+                 |-----------------|1--+
          |--------x--------|2--|+                |-----------------|3--|-+
          |-----------------|3--||+               |-----------------|4--|-|+
          |-----------------|4--|||+    FreePtr-> |-----------------|5--|-||+
FreePtr-> |-----------------|5--||||+             |        *        |   |  ||
          |        *        |   |||||             |                 |   |  ||
          |                 |   |||||             |                 |   |  ||
          |                 |   |||||             |                 |   |  ||
          |                 |   |||||             |                 |   |  ||
          |        *        |   |||||             |        *        |   |  ||
HeapPtr-> |-----------------|   |||||   HeapPtr-> |-----------------|   |  ||
          |+++++++++++++++++|   |||||             |+++++++++++++++++|   |  ||
          |+++++++++++++++++|1<-+||||             |+++++++++++++++++|1<-+  ||
    (A)>>>|+++++++++++++++++|    ||||             |                 |      ||
          |+++++++++++++++++|2<--+|||       (A)>>>|-----------------|      ||
          |+++++++++++++++++|     |||             |+++++++++++++++++|      ||
          |+++++++++++++++++|3<---+||             |+++++++++++++++++|3<---+||
          |+++++++++++++++++|      ||             |+++++++++++++++++|       ||
          |+++++++++++++++++|4<----+|             |+++++++++++++++++|4<----+|
          |+++++++++++++++++|       |             |+++++++++++++++++|       |
          |+++++++++++++++++|5<-----+             |+++++++++++++++++|5<-----+
          |    * Program *  |                     |    * Program *  |
```

If adjacent memory is found, the free space pointer is removed from the FreePtr chain and TEGL's heap manager moves the free chain structure up by one entry to close the empty position. The size and the original

pointer (A) is adjusted to reflect a new pointer position and size.

```
           Top of DOS Memory                      Top of DOS Memory
          +-----------------+                    +-----------------+
          |-----------------|3----+              |-----------------|A--+
          |-----------------|4----|+             |-----------------|3--|-+
FreePtr-> |-----------------|5----||+            |-----------------|4--|-|+
          |        *        |     |||  FreePtr-> |-----------------|5--|-||+
          |                 |     |||            |                 |   | |||
          |                 |     |||            |                 |   | |||
          |                 |     |||            |                 |   | |||
          |                 |     |||            |                 |   | |||
          |                 |     |||            |                 |   | |||
          |        *        |     |||            |        *        |   | |||
HeapPtr-> |-----------------|     |||  HeapPtr-> |-----------------|   | |||
          |-----------------|     |||            |+++++++++++++++++|   | |||
          |                 |     |||            |||||||||||||||||||   | |||
          |                 |     |||            |||||||||||||||||||   | |||
    (A)>>>|-----------------|     |||            |+++++++++++++++++|A<-+ |||
          |+++++++++++++++++|     |||            |+++++++++++++++++|     |||
          |+++++++++++++++++|3<---+||            |+++++++++++++++++|3<---+||
          |+++++++++++++++++|      ||            |+++++++++++++++++|      ||
          |+++++++++++++++++|4<----+|            |+++++++++++++++++|4<----+|
          |+++++++++++++++++|       |            |+++++++++++++++++|       |
          |+++++++++++++++++|5<-----+            |+++++++++++++++++|5<-----+
          |   * Program *   |                    |   * Program *   |
```

When all possible adjacent blocks have been removed, the TEGL heap manager checks if the end of memory block is equal to the HeapPtr. If not, a free space entry is added to the bottom of FreePtr.

FREEMEM(B)

```
           Top of DOS Memory                      Top of DOS Memory
          +-----------------+                    +-----------------+
          |-----------------|A--+                |-----------------|3----+
          |-----------------|3--|-+              |-----------------|4----|+
          |-----------------|4--|-|+   FreePtr-> |-----------------|5----||+
FreePtr-> |--------x--------|5--|-||+            |        *        |     |||
          |        *        |   | |||            |                 |     |||
          |                 |   | |||            |                 |     |||
          |                 |   | |||            |                 |     |||
          |                 |   | |||            |                 |     |||
          |                 |   | |||            |                 |     |||
          |        *        |   | |||            |                 |     |||
HeapPtr-> |-----------------|   | |||            |                 |     |||
    (B)>>>|-----------------|   | |||            |                 |     |||
          |                 |   | |||            |                 |     |||
          |                 |   | |||            |        *        |     |||
          |-----------------|A<-+ |||  HeapPtr-> |-----------------|     |||
          |+++++++++++++++++|     |||            |+++++++++++++++++|     |||
          |+++++++++++++++++|3<---+||            |+++++++++++++++++|3<---+||
```

```
          |+++++++++++++++++|      ||          |+++++++++++++++++|      ||
          |+++++++++++++++++|4<----+|          |+++++++++++++++++|4<----+|
          |+++++++++++++++++|       |          |+++++++++++++++++|       |
          |+++++++++++++++++|5<-----+          |+++++++++++++++++|5<-----+
          |   * Program *   |                  |   * Program *   |
```

TEGL uses the more efficient method of maintaining the free space chain in sorted out. This allows allocation of memory to favor the lower portion of the heap. However, as mentioned before, this does not remove the fragmentation problem where one non-movable records is allocated in the middle of the heap.

Combining the best of both Heap Managers (Coexisting)

What we noted that we needed was the ability to have two heaps. One for miscellaneous dyanamic variables and one for large allocations for images. Combined with the virtual memory handler, this allows the paging out the large allocations effectively releasing adjacent memory. At the same time we did not want to limit either heap. The turbo heap must have the ability to flow over to the second heap without problems.

ReserveHugeMinimum provides an elegant solution of partitioning the standard heap into two parts. A single non-movable byte is allocated as a partitioner.

```
                                      GetMem                  TEGLGetMem
          +-------+                +-------+                +-------+
FreePtr-> |-------|1--+  FreePtr-> |-------|1--+  FreePtr-> |-------|1--+
          |       |   |            |       |   |            |       |   |
          |       |   |            |       |   |            |       |   |
          |       |   |            |       |   |            |       |   |
          |  329k |   |            |  329k |   |            |  200k |   |
          |       |   |            |       |   |            |       |   |
          |       |   |            |       |   |  HeapPtr-> |-------|   |
          |       |   |            |       |   |            |///////|   |
          |       |   |            |       |   |            |///////|   |
          |HugeMin|   |            |HugeMin|   |            |///////|   |
HeapPtr-> |-------|   |  HeapPtr-> |-------|   |            |-------|   |
          |       |   |            |       |   |            |       |   |
          |  12k  |   |            |  11k  |   |            |  11k  |   |
          |       |   |            |       |   |            |       |   |
          |       |   |            |       |   |            |       |   |
          |       |   |            |-------|1<-+            |-------|1<-+
          |       |   |            |///////|                |///////|
          |-------|1<-+            |-------|                |-------|
          |Program|                |Program|                |Program|
```

Since Turbo Heap Manager will always search for free space through the FreePtr Chain, the lower partitioned area will always be used first (it

is always the first few entries in the FreePtr chain). (Remember, when Turbo Pascal frees a memory block, the free space pointer will be the most recent entry.)

The TEGL heap manager will always attempt to allocate space between HeapPtr and FreePtr before searching through the free space pointer chain. Even when searching through the free space chain, a comparison is made on the minimum area for allocating. When TEGL frees a memory block, the free space pointer is sorted upwards into the free space chain.

```
           Top of DOS Memory
          +-----------------+
      +--A|-----------------|
     +|--B|-----------------|
    +||--C|-----------------|
   +|||--D|-----------------|
  +||||--E|-----------------|
  |||||   |-----------------|4--+
  |||||   |-----------------|1--|+
  |||||   |-----------------|3--||+
  |||||   |-----------------|2--|||+
  |||||   |-----------------|5--||||+
  |||||   |                 |   |||||
  ||||+->A|+++++++++++++++++|   |||||
  |||+-->B|+++++++++++++++++|   |||||
  ||+--->C|+++++++++++++++++|   |||||
  |+---->D|+++++++++++++++++|   |||||
  +----->E|+++++++++++++++++|   |||||
          |    Hugemin      |   |||||
          |-----------------|   |||||
          |+++++++++++++++++|2<-|+|||
          |+++++++++++++++++|3<-|-+||
          |+++++++++++++++++|5<-|--|+
          |+++++++++++++++++|4<-+  |
          |+++++++++++++++++|2<----+
          |   * Program *   |
```

Appendix C - Vars, Types & Const

ActivePage Word Typed Const FASTGRPH

Set to the memory address (segment) of the active video page.

See also FlipAPage, FlipVPage, SetAPage, SetVPage.

CallProc Procedure Type TEGLUNIT

This is the standard declaration for and event. All procedures and functions that set events specify this in their parameter list.

See also (it NilUnitProc.

FG* Const FASTGRPH

These constants are used as an arguments to PutBiti and can be assigned to RmwBits. Determines what kind of binary operation to use. They are:
FGNorm - Normal or Copy put, FGAnd - AND put, FGOr - OR put,
FBXor - XOR put, FGNot - not put.

See also FastLine.

Jagged Word Typed Const FASTGRPH

This const affects all output by OutTEGLTextXY and TEGLWrtChar. When set to 0 no action is taken, when set to 1, text is output with alternate rows of pixels shifted by one.

MSClick Boolean Const TEGLUNIT

Set to False. Used as an argument to mouse related procedures where mouse activation is desired by location over the mouse click area and pressing the left button.

See also DefineMouseClickArea, ResetMSClickSense.

MSSense Boolean Const TEGLUNIT

Set to True. Used as an argument to mouse related procedures where mouse activation is desired by simply passing over a mouse click area.

See also DefineMouseClickArea, ResetMSClickSense.

RmwBits Word Typed Const FASTGRPH

Set to the desired binary operation for subsequent PutPixs.

See also FG*.

The file switches.inc contains conditions compilation directives that support different facilities with the Toolkit.

Note that if you change any defines you will have to make the entire toolkit.

The following defines affect the Toolkit:

{$DEFINE AllFonts} - The toolkit is built referencing the unit {it TeglFont} for getting the address of a font. If this symbol is not defined then the fonts are referenced in seperate units. Having the fonts in individual units has the advantage of saving some memory space during linking (assuming they are not all used). If you are using the integrated environment then commonly used fonts can be loaded directly into turbo.tpl for faster compiles.
Font units file names are of the form fon*.pas.

{$DEFINE NoGr} - The toolkit is built with no explicit references to the Graph unit provided with Turbo Pascal. Instead a compatible unit

Conditional Compilation

TGraph is uses which provides a subset of the functions provided in Graph. If your application does not need all the features of the Graph unit then compiling with this directive enabled can save as much as 25K of code size in a program (assuming the BGI drivers are linked in).

{$DEFINE NoVirt} - The code that implements virtual memory using either EMS or a disk drive is not included. Applications save about 8K of code space but can easily run out of memory if many windows are opened. This is more critical for EGA or VGA displays since the windows require four times as much memory than CGA or Hurcules displays.

{$DEFINE Quick} - The tookit will be built using the {it MSGraph} unit provided with Quick Pascal. TGraph is used to map calls to the appropriate routines in MSGraph.

These defines determine what graphics boards will be supported. The assembly language code that implements the drivers for each board takes about 3K of space in the final application. You cannot define all of these.

{$DEFINE NoCGA} - The code for the color graphics adaptor is not linked.

{$DEFINE NoEVGA} - The code for the enhanced graphics adaptor and the video graphics array is not linked in.

{$DEFINE NoHerc} - The code for the hercules graphics adaptor is not linked in.

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX