EllTech

Graphical User Interface Toolkit

Reference Manual

"EGUI"

Dedication,

The creation of the EGUI Toolkit Product was only possible because of the best family a guy could have. The hundreds of hours which has gone into developing this product over a five year period have been hours that could have been spent with my family. But the hard work put into this product was done because of my family. I would like to say Thank You and that I Love each one of you very much, especially my beautiful wife Lisa.

Thanks Guys, I couldn't have done it with out you!

Mike

Developed for **EllTech Development, Inc.**

4374 Shallowford Industrial Parkway
Marietta, GA 30066

Technical Support	(404) 928-8960	9am - 5am M-F
Sales	(800) 553-1327	
BBS	(404) 928-7111 (HST)	
Fax	(404) 924-2807	

Documentation Revision Date: January 1993

Contents

Part 1 Getting Started

Chapter 1 Introduction

Chapter 2 Set Up (Installation)

Part 2 EGUI Form Generator

Chapter 3 Overview

Chapter 4 Drawing the Interface

Chapter 5 Setting Properties

Chapter 6 Creating Form Source Code

Chapter 7 Printing

Part 3 Application Development

Chapter 8 Structuring the Application

Chapter 9 Attaching Code to Controls

Part 4 Controls, Properties & Functions

Chapter 10 Environment Properties

Chapter 11 Control Properties

Chapter 12 Control Functions

Chapter 13 Library Functions

Chapter 13 Continued

Appendix

EGUI Toolkit
Part 1

Getting Started

Document Conventions

Throughout this manual, the term "DOS" refers to both MS-DOS and PC-DOS.

The following document conventions are used throughout this manual:

Convention	Description of Convention
Bold text	Bold letters indicate a specific term or punctuation mark intended to be used literally; (i.e. language key words or function names such as **REDIM** or **gBuildDialogBox%**). You must type these terms and punctuation marks exactly as shown.
( )	In syntax statements, parentheses enclose one or more parameters that you pass to a function.
Italic text	Words in italics indicate a placeholder; you are expected to provide the actual value. For example, the following syntax for the **gPaint%** function indicates that you must substitute values for the *x1%*, *y1%* and *colr&* parameters, separated by a comma: **gPaint%**(*x1%, y1%, colr&*)
`Monospaced type`	Code examples are displayed in a nonproportional typeface.
BEGIN that a . . . END	Vertical ellipses in program example indicate portion of the program is omitted.
...	Ellipses following an item indicate that more items having the same form may appear.

<table>
<tr><th>Convention</th><th>Description of Convention</th></tr>
<tr><td>[ ]</td><td>Brackets enclose optional fields or parameters in command lines and syntax statements. In the following example STATIC is an option in the function header.
<b>FUNCTION glnitBWSystem% [STATIC]</b></td></tr>
<tr><td>|</td><td>A vertical bar indicates that you may enter one of the entries shown on either side of the bar. The following statement illustrates the use of a vertical bar:
colr& = defaultflag | colorformula</td></tr>
<tr><td>" "</td><td>Quotation marks set off terms defined in the text.</td></tr>
<tr><td>SMALL CAP LETTERS</td><td>Small capital letters indicate the names of keys and key sequences, such as:
ALT + SPACEBAR</td></tr>
</table>

Introduction

Chapter 1

Product Description

Welcome to the EGUI Toolkit Library and Form Generator! We have put every effort into making this one of the finest Graphical User Interface Libraries available to programmers.

EGUI Toolkit is a library of BASIC and Assembly Language routines that are Linked into your .EXE program files or placed in an extended runtime library.

The EGUI Toolkit was developed to allow the building of DOS based application with a Graphical User Interface. These applications would be able to function much like applications which are developed for Microsoft Windows, but without the overhead involved with MS Windows programs.

The EGUI Form Generator was developed to allow the creation of Dialog Box Forms which are used within the EGUI Library to build the application's user interface. The Form Generator works much like that of MS Visual BASIC. You draw Control Objects on a Dialog Box Form. Then you may customize the forms and controls to you specific needs by setting their property values. After a form has been created you may save a form file to be modify later or create the BASIC Source Code to load into the BASIC IDE System or your favorite Text Editor.

The EGUI Source Code Compiler (Generator) will read and interpret EGUI Form Files and produce BASIC Run Time Source Code. These Source Code Files are either full working modules, or function files which may be merged into a module. The files are produced in ASCII code so they are compatible with a wide variety of Text Editors. Once the Source Code Files have been loaded into your editor then you will attach code to the events which are produced by the User Interface Library. This code will add the finial functionality to you applications.

The EGUI Library is based on Object Orient Programming Techniques. It is not a true Object Orient Library, however it includes, and will allow the development of, many of the features and capabilities found in an Object Orient System. This manual does not go deep into the use of Object Orient Techniques because that is beyond its scope. However it is recommend, but not required, that if you are not familiar with Object Orient Programming Techniques that you should acquire a reference manual on this subject. There are several excellent reference available on Object Orient Programming. If you can not find a reference we will be glad to may some suggestions.

Compatible Compilers

EllTech Graphics User Interface Toolkit is compatible with Microsoft QuickBASIC versions 4.00b and 4.50, Microsoft BASIC Compiler 6.x, Microsoft BASIC Professional Development System 7.x (PDS) and Visual Basic for MS-DOS version 1.0 (VBDOS). The QuickBASIC, PDS, VBDOS versions are each sold separately.

System Requirements

◆ During Development

EGUI Form Generator & EGUISGEN (Source Code Compiler)

IBM PC or PS/2 or 100% Compatible
286 CPU or better [386 recommended]
VGA Video Adapter
DOS 3.1 or higher
560k Conventional Memory
[1.4 meg of EMS or XMS recommended]
Hard Drive with 2 meg available
[1meg Ramdisk highly recommended]
Microsoft Mouse or 100% Compatible

The EGUI Form Generator will use EMS or XMS memory for some of its code if avalible. The EGUI System uses a Ramdisk or Harddrive for graphics screen paging and data storage. For best performance setup a 1meg Ramdisk (**Note:** Put the environment variable RAMDISK=drive:\ in your AUTOEXEC.BAT).

Example: **SET RAMDISK=G:\** *(where G is your ramdisk drive id)*

Note any other utilities which come with EGUI will work with the above requirements.

◆ During Run Time

IBM PC or PS/2 or 100% Compatible
286 CPU or better [386 recommended]
256k EGA or VGA Video Adapter (SDVM)
DOS 3.1 or higher
180k Conventional Memory
[1 meg of EMS or XMS recommended]
Hard Drive with 1.4 meg available
[1meg Ramdisk highly recommended]
Microsoft Mouse or 100% Compatible

In addition to the above requirements you must add your application overhead to the Ram and Harddrive requirements. Also not that the 256k video requirements are for the EGUI Standard Video Driver Module, if an Enhanced Video Driver Module *(this is a sperate purchase item)* is used see the requirements for that specific driver.

Product Registration

If EGUI Toolkit was purchased directly from EllTech Development, it has already been registered to the person who bought it. If it was purchased through a dealer, your registration information will be forwarded to EllTech Development by your dealer. Upon receiving this information, you will be added to our customer database and will be given access to the EGUI Toolkit Support Conference on our BBS.

Technical Support

EllTech Development provides free, full-time technical support to all registered users of EGUI Toolkit. Our hours are Monday through Friday, 9:00 a.m. to 5:00 p.m. Eastern time. You can reach us at **(404) 928-8960**.

Or if you prefer, call our 24 hour bulletin board system ("BBS"). We run PCBoard BBS software and a US. Robotics Courier Dual Standard HST modem (supporting baud rates from 1200 to 38400, including v.32 high-speed modems). We have a dedicated message base for EGUI Toolkit as well as the latest version of the product available for download. Many times you can get an answer to your tech support question by calling our BBS and scanning the messages. If you've run into a snag, the chances are pretty good that others have had a similar problem and a solution already awaits you.

As a registered user of EGUI Toolkit, you already have an account established on the BBS as well as access to the private EGUI Toolkit conference. Log on using the name that appears on your invoice (no middle initials). Your password is your Zip or Postal code. Be sure to use the BBS's "W" command to change your password (for security reasons) during your first session.

Here are some phone numbers you'll need to know:

(404) 928-8960	Technical Support
(404) 928-7111	EllTech Development's BBS
(404) 924-2807	Fax

Set Up (Installation)

Chapter 2

Getting Started

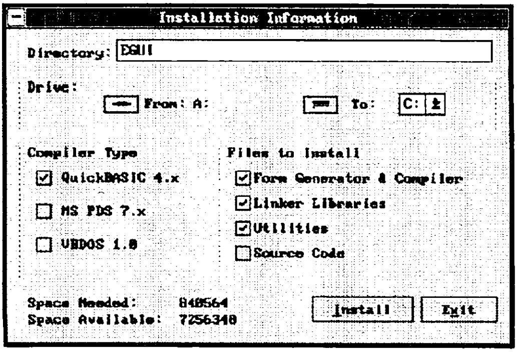
◆ Making Backup Copies

Before installing EGUI Toolkit for the first time, be sure to make a backup copy of the distribution diskette(s) for safe keeping. The DOS "DISKCOPY" command is best suited for this purpose. If you require any help using DISKCOPY, please refer to your DOS reference manual.

◆ Installation

To install EGUI Toolkit on your hard disk drive, insert the distribution diskette into the appropriate floppy disk drive and close the latch. Log onto that floppy drive by typing the drive letter followed by a colon and press ENTER. Next, you'll need to run the "INSTALL.EXE" program. This will decompress and copy the appropriate files to the specified subdirectory on your hard drive. This is the same installation program that is included as part of this product. It is easy to set up and customize for your own use.

Once you execute INSTALL.EXE, click the INSTALL button on the startup screen and you will be asked to provide information such as the desired drive and directory that you wish to install EGUI Toolkit on, the compiler(s) that you are using, and the files that you wish to install. Once all of the required information is provided, click the INSTALL button and INSTALL will automatically create the required directories and decompress the required files.



Directory: EGUI

Drive:
From: A:
To: C:

Compiler Type
☑ QuickBASIC 4.x
□ MS PDS 7.x
□ VBDOS 1.0

Files to Install
☑ Form Generator & Compiler
☑ Linker Libraries
☑ Utilities
□ Source Code

Space Needed: 840564
Space Available: 7256340

Files Distributed with EGUI Toolkit

After installing EGUI Toolkit onto your hard drive, the following files and subdirectories will be present in and under the default EGUI Toolkit directory:

◆ \EGUI Default Directory

EGUI.EXE The EGUI Form Generator program.

EGUISGEN.EXE The EGUI Source Code Compiler (Generator). This program is used to build source code files (*.BAS and *.FNC) from form files (*.FRM).

EGUI.INI The EGUI System Initialization File. This file is used with the Form Generator and must also be included with any applications you create with the EGUI System Library.

README.DOC If present, it contains important information such as documentation errata or other additions or corrections made to EGUI Toolkit after the manual went to press.

EGUIMAIN.PCX Form Generators startup screen. This screen may be disable in the EGUI.INI files.

EGUIABT.PCX Form Generators About Box Icon.

◆ \LIB Linker Libraries & Object Modules

EGUIQB45.LIB The EGUI Linker Library for QuickBASIC 4.00b and 4.5.

EGUIBC7F.LIB The EGUI **Far String** Linker Library for Microsoft PDS BASIC 7.x.

EGUIBC7N.LIB The EGUI **Near String** Linker Library for Microsoft PDS BASIC 7.x.

EGUIVBD.LIB The EGUI Linker Library for Microsoft Visual BASIC for DOS 1.0.

SVDMQ45.OBJ The EGUI Standard Video Driver Module Object File for QuickBASIC 4.00b and 4.5.

SVDMB7F.OBJ The EGUI **Far String** Standard Video Driver Module Object File for Microsoft PDS 7.x.

SVDMB7N.OBJ The EGUI **Near String** Standard Video Driver Module Object File for Microsoft PDS 7.x.

SVDMVBD.OBJ The EGUI Standard Video Driver Module Object File for Microsoft Visual BASIC for DOS 1.0.

◆ \INCLUDE EGUI Include Files

BWCTR.INC The EGUI Control and Procedure Declaration File. This file must be include in any module making calls to the EGUI Library.

BWENV.INC The EGUI Environment Property File. This file must be include in any module making calls to the EGUI Library.

BWPRP.INC The EGUI Control Property and Global Constant and Variables File. This file must be include in any module making calls to the EGUI Library.

◆ \ICON The EGUI Icon Files

There are more than fifty EGUI Icon files which are used with the EGUI System and Form Generator. There are also some icons which are used with sample programs. The icon file names starting with the letters "BWC" all are used with EGUI Form Generator. The icon files listed below are used with the EGUI System and must be included with any applications you create.

BWCMB.ICN ** **BWDNARW.ICN** **BWEROR.ICN**
BWINFO.ICN **BWLFARW.ICN**
BWRTARW.ICN
BWSTOP.ICN **BWUPARW.ICN** **BWWARN.ICN**

You may use and distribute icon files in any manner you desire.

◆ \UTILITY EGUI Utility Programs

CM.COM

The Compiler Manager Program. This program may be used to manage the building of your application during development. See **Appendix C Utilities** for more information.

CMMAIN.CFG

The Compiler Manager Main Configuration File.

CM.CFG

The Compiler Manager Local Configuration File.

PCX2ICN.EXE

A conversion program used to convert .PCX image files to .ICN image files. See **Appendix C Utilities** for information on how to use this program.

◆ \SOURCE EGUI System Source Code

BWMS1.BAS

EGUI System Library Module One.

BWMS2.BAS

EGUI System Library Module Two.

- ### \SAMPLE EGUI Sample Programs

There are several directories off the Sample Directory which have sample Form, Function and Modules files. Read the README.DOC file in each of these directories to get more information on that sample program.

- ### \TMP & \HELP & \DRV EGUI System Directories

The System Temporary Directory (\TMP) is used to store temporary files which the EGUI System creates during Run Time. This directory must be off your application's root directory and pointed to in the EGUI.INI file for proper system operation.

The Help and Driver Directories are for future development, but they should be included with any applications you create.

Procedure Organization

The EGUI Toolkit routines are in two groups, the **Control Functions** and **Library Functions**. The Control Functions are what you will use most to create your applications. In fact when you build a form with the Form Generator and produce its source code, you will find that most of the routines used in this source code are all listed in the Control Functions Chapter. The Library Functions are a group of high-level and low-level routines which should be used for developing your application once you begin attaching code.

Important: Make sure to use the procedures in the Library Functions Chapter which replace BASIC's primitive drawing functions. Such as gDrawLine% instead of LINE and gDrawCircle instead of CIRCLE, etc., because these procedures follow the EGUI Video Driver Specification. This will allow you to change Video Driver Modules and get increased performance and features with minimum changes to your application.

The other chapters which are very important are Chapters 10 & 11 (Environment and Control Properties). The properties in this system are as important as the procedures are. They are used much like nested procedures would be use to control the over all procedures characteristics.

Procedure Declarations (Include Files)

Most of the EGUI Toolkit procedures receive their arguments "by reference", "by value" or by a "segmented address." We have defined the correct parameter-passing conventions for EGUI Toolkit routines using BASIC's "DECLARE" statement. The declarations for the *Control Functions* can be found in the file called "**BWCTR.INC**". As long as you **$INCLUDE** this file at the top of each program source file that invokes a EGUI Toolkit procedure, you will not have to give parameter-passing conventions another thought.

There are two other include files which are very important also, "**BWENV.INC**" and "**BWPRP.INC**". These include files define and allocate memory for the EGUI System Control Property Structures and Global Constants and Variables. These files should also be included in your application module code with the **$INCLUDE** statement. These files will add some overhead to you .EXE file, but they must be present for proper operation.

Important: All three Include Files <u>must</u> be include in your applications for proper operation of the library.

Using PDS's and VBDOS "Far String" Option

If you installed EGUI Toolkit for the Microsoft PDS compiler, two LINK libraries were installed.

◆ **EGUIBC7F.LIB**	**For "Far String" versions of the routines.**
◆ **EGUIBC7N.LIB**	**For "Near String" versions of the routines.**

LINK to **EGUIBC7F.LIB** when compiling with PDS's "/FS" switch and LINK to **EGUIBC7N.LIB** when compiling without "/FS." Although the far string option does make more variable-length string space available, your programs' performance will suffer slightly due to the complex nature of BASIC's far string management code.

If you installed EGUI Toolkit for Microsoft Visual BASIC for DOS the following LINK library was created.

- **EGUIVBD.LIB**

If you installed EGUI Toolkit for QuickBASIC 4.x or BASIC 6.x, only one LINK library was created.

- EGUIQB45.LIB

Building Quick Libraries

To build Quick Libraries from the Linker Libraries which where created at installation time move to your EGUI Default Directory and run the BLDQLB.BAT file with one of the following switches.

/QB45	- Build QuickBASIC 4.x Quick Library
/BC7	- Build MS PDS BASIC 7.x Quick Library "Far Strings"
/VBD	- Build MS Visual BASIC for DOS 1.0 Quick Library

Example *(shown for QB 4.5)*:

```
C:\EGUI>BLDQLB /QB45
```

Make sure that the correct version of LINK.EXE and also the Quick Library for the compiler version you are building for, is in your DOS Environment Path, or the build process will not function properly.

Compiling and LINKing

When "Making an EXE" from within the QB(x) or VBDOS environment, the resulting .EXE file is usually larger than necessary because the IDE often includes compiler switches which you do not need. For this reason, we recommend that you compile and LINK your program manually, from the DOS prompt. This gives you absolute control over what goes into your .EXE file (at least all of the control that Microsoft gives you). It only involves two steps:

Compile your program's source code files. This is accomplished by using the BC.EXE program. For example:

```
BC PROGRAM [switches] ;
```

"Switches" might include "/O" to make your program a stand-alone, "/FS" to take advantage of PDS's "far string" option, etc. Consult your compiler documentation for a complete list of available options. For example:

```
BC PROGRAM /O ;
```

This would compile PROGRAM.BAS using the "stand-alone" option, generating PROGRAM.OBJ.

The next step is to LINK the PROGRAM.OBJ file with all of the other code required to make it an executable program. Such "support code" is usually found in libraries, such as BCOMxx.LIB (QuickBASIC), BCL71ENR.LIB (PDS), etc. For example:

```
LINK [switches] PROGRAM, , NUL, [libraries] ;
```

"Switches" usually include "/EX" which compresses your .EXE file by five to forty percent. PROGRAM is the name of the .OBJ file generated by the compiler in the first step. The two commas that follow the program name direct LINK to give the resulting .EXE file the same primary name as the first .OBJ listed. In this case, the .EXE file will be called PROGRAM.EXE. The "NUL" directs the LINK program not to generate a "map" file (they're of little use to BASIC programmers). Finally, you can list one or more entries in the "libraries" field. By default, the LINK program automatically searches the compiler support libraries, so there's no need to list them. If you also wish LINK to look in other libraries as well, such as EGUI Toolkit's library, you can list it here. For example:

```
LINK /EX PROGRAM, , NUL, EGUIQB45 ;
```

This would LINK PROGRAM.OBJ into an .EXE file. All routines required by PROGRAM that are present in the BASIC support library and the EGUI Toolkit library would be brought into the .EXE file automatically. Again, only code required by PROGRAM will be extracted from the library(s) and made part of the final .EXE file. Assuming there were no LINKer errors, PROGRAM.EXE would be generated and ready to execute at the completion of this step.

EGUI Toolkit
Part 2

EGUI Form Generator

Overview

Chapter 3

What is an Object?

An **Object** is anything that a user can manipulate as a single entity. Objects are always the focus of the user's attention.

Control Objects are the primary objects used in the EGUI System. These are Graphical Entities, such as an Edit Box or Command Button, which are placed on a Dialog Box Form. Control Objects are use to get input from the user or display output. Each Control Object has its own set of recognized properties which allow the manipulation of that Object by the user.

Below is a list of supported Control Objects in the EGUI System. See Chapter 12 for a definition of each of these Control Functions.

Control	Function Name
Dialog Box	gBuildDialogBox%
Picture Box	gDrawPCXFile% **
Label	gDrawTextPix% **
Edit Box	gEditBox%
Frame Box	gDrawRect% **
Command Button	gCommandButton%
Check Box	gChkOptBox%
Option Button	gChkOptBox%
Combo Box	gComboBox%
List Box	gListBox%
Horizontal Scroll Bar	gHorzScrollBar%
Vertical Scroll Bar	gVertScrollBar%

** These Functions are not Controls they are procedures in the EGUI Library. The EGUI Form Generator uses these procedures to emulate a control process. When the source code is produced by the EGUI Source Code Generator these procedures will be used. See Chapter 13 for the definitions on these procedures.

A Dialog Box is also a Control Object however it is referred to as a Form at Design Time. Anytime a Form is mentioned in this manual it is referring to a Dialog Box.

During Design Time the Control Objects may be selected from the Control Menu or from the Control Toolbox. See Chapter 4 Drawing the Interface for more information.

Design Mode & Run Time Mode

Version 1.0 of the EGUI Form Generator will only run in **Design Mode**. After a form has been designed, use the **Create Form Function** from the File Menu to build a fully functional **BASIC** Module or Function that may be loaded into the IDE System or any Text Editor or Compiled. Note that even though the Form will run after source code has been produced by the EGUI Source Code Generator *(also called the EGUI Source Compiler)*, it is still necessary to attach code to the form's events and data structures before the form is fully functional.

Future versions of this software will have the capability to run in both **Design & Run Time Modes** and also allow the attachment of some code to a form within the Generator. This should allow the full development of small application within the Generator itself.

Working with Forms (Dialog Boxes)

When starting the EGUI Form Generator a Dialog Box Form is automatically created and placed in the center of the display. The form may be moved or resized as needed while in Design Mode *(see Resizing, Moving & Deleting Controls in Chapter 4 for more information on moving and resizing forms).*

The EGUI Form *(Dialog Box)* is the foundation for the EGUI System. Forms allow the user to focus on a specific topic, or a group of related topics, while isolating the user from other portions of the application. This is one of the key factors of an Object Oriented Application.

You may only load one form into the Generator at a time. So if you are working with multiple forms you must save your work and then load or create a new form.

◆ Saving a Form

To Save a Form select **Save Form** or **Save Form As** from the File Menu. If Save Form is selected and you are working with a form that has an UNTITLED.FRM name the Save Form As Dialog Box will appear to prompt for a new name. You may save the form as UNTITLED.FRM, but this is not recommended because each time you save the form this prompt will appear. When the Save Form As Dialog Box appears enter a valid DOS filename with the extension **.FRM** added to the form name or select a previous form name from the Files List Box. You may also select a new directory or drive. Then click the OK button to save the file. Each time the Save Form process is invoked after this it will simply save the file. You may also create a copy of a form by doing a Save Form As at any time.

◆ Creating a New Form

To create a new Form select **New Form** from the File Menu. A Dialog Box will open allowing you to save the current form before creating the new one. Select **Yes** to save the current form or **No** to discard the current form. You may also click the Close Button to Cancel the process.

◆ Loading a Form

To load an existing form select **Load Form** for the File Menu. When the Load Form Dialog Box opens select the form file you wish to load from the File List Box. You may optionally move to a new directory or drive to select your file. Click the OK button and the form will be loaded.

Form files are saved and loaded from the path location specified during the initial save or load process. For additional information on forms see Chapter 4 Drawing the Interface.

Color System

The EGUI System is based on True Color Technology. This will allow for future EGUI Video Driver Modules to support extended Video Color Systems and Resolutions. This Version of the EGUI Library comes with the **EGUI Standard Video Driver Module (EGUI SVDM).** This module supports display modes 9 and 12 for EGA/VGA video adapters with 16 colors. For most application one of these modes should be sufficient. It is highly recommended to use mode 12 standard VGA 16 color if at all possible. The best overall performance and features will be obtained when using this mode. Also this has become the industry standard for graphical application. **Important:** When using the Standard Video Driver Module with an EGA Video Adapter, the adapter must have at least 256k of video memory. This module is **NOT** compatible with a 64k EGA Video Adapter.

There are 16 EGUI Environment Color Properties which should be used to assign any color attributes in your application. By using these properties to assign color attributes it will dramatically improve the capability of incorporating future enhancement of the EGUI System. This will also make it easier to add new Video Driver Modules with very little, if any, modification to your existing code.

◆ Environment Color Properties

Property	EGUI SVDM Attribute Value
bwEV(0).Black	0
bwEV(0).Brown	1
bwEV(0).Green	2
bwEV(0).Orange	3
bwEV(0).Blue	4
bwEV(0).Magenta	5
bwEV(0).Cyan	6
bwEV(0).DarkGray	7
bwEV(0).Gray	8
bwEV(0).Red	9
bwEV(0).LightGreen	10
bwEV(0).Yellow	11
bwEV(0).LightBlue	12
bwEV(0).LightMagenta	13
bwEV(0).LightCyan	14
bwEV(0).White	15

These properties are initiated when you initialize the EGUI System *(See Chapter 8 Structuring the Application for more information).*

The EGUI System uses two color formulas to assign color attributes to controls and function procedures in the Library. Below are definition of these formulas.

◆ Single Color Formula

The Single Color Formula is nothing more than assigning a single Environment Color Property to any Control Color Attribute Property or Library Function Procedure argument. Note that this assignment may effect the foreground or background information of the object it is assigned to. You should refer to the documentation on the specific Control or Function Procedure to determine the effect of this assignment. Below is an example of the use of this formula. The environment color cyan is being assigned to the background color of a Dialog Box.

```
bwDB(0).dcolr = bwEV(0).Cyan
```

◆ Multi Color Formula

The Multi Color Formula is the simultaneous assignment of Two Environment Color Properties to any Control Color Attribute Property or Library Function Procedure argument. The two color attribute are passed to the Control or Function Procedure by one value. The combing of the two attributes is done by multiplying second attribute value by the value 256 and then adding the first attribute value to this value. Usually the first color assignment effects the foreground information of an object and the second color assignment effects the background information of the object. There are some exception to this rule but they will be documented where needed. Below is an example of the use of this formula. The environment color Blue is being assigned to the foreground text color of a Command Button and the color Gray is being assigned to the background.

```
bwBT(0).dcolr = (bwEV(0).Blue + (bwEV(0).Gray * 256))
```

System Color Palette Assignment

The EGUI System Color Palette is readjusted at initialization time to the True Color Palette Settings. Note that this palette information is controlled internally by the system and may not be adjusted.

Important: One exception to this is the colors Gray an Dark Gray maintain the Standard IBM PC Color Palette Settings. This is because of the use of these colors for background information on Command Buttons.

Important: It is recommended to use the default system palette settings however, if you wish for your application to maintain the Standard IBM PC Palette Settings set the Environment Property bwEV(0).Setpaletteflag to False (0). This may be done in the EGUI.INI file by make the following assignment.

```
Setpaletteflag = 0
```

Note that one of the primary purposes of the EGUI System is to allow the co-development of application in both a Microsoft Windows and DOS Environment. So by using the default system palette settings any .PCX files or converted .ICN files will maintain a similar appearance in a DOS application as the Windows version. Also standard Dialog Boxes and Controls maintain this appearance.

Desktop Color & Background

When the EGUI System is initialized it will display a Desktop Background by filling the current display screen with a specific color or loading a .PCX image file. The settings to control this process are located in the EGUI.INI file. Use the settings listed below to control this process.

To assign a Desktop background Color:

```
Desktopcolr = 12        'EGUI SVDM LightBlue Attribute
                        'This may be any value (0-15)
                        '12 is the Default Setting
```

To assign a Desktop background PCX File:

```
DeskTopFile = EGUIDEMO.PCX      'Desktopcolr property
                                'is ignored
```

To turn off the Desktop Process:

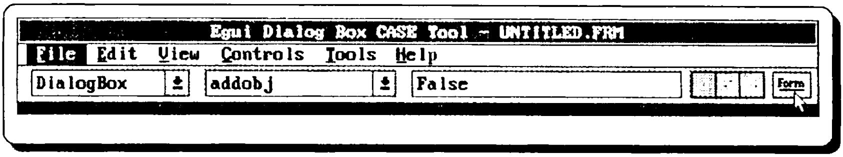
```
Desktopcolr = -1            'Set a -1 Value
```

Note the PCX file must be a 16 color PCX file format. And preferably should cover the entire display.

Full Form Mode

By default the EGUI Form Generator is set to **Standard Form Mode** which will allow the development of a Dialog Box Form about the size of 3/4 the display. This is more than sufficient for most Dialog Box development. However occasionally you may need to use the full display to build a Dialog Box.

To enter **Full Form Mode** select **Form Full Screen** from the View Menu. After setting this mode active the FORM button on the Control Property Tool Bar will be enabled, you must click this button to edit a full form Dialog Box.


Egui Dialog Box CASE Tool - UNTITLED.FRM

DialogBox ± | addobj ± | False

Control Property Tool Bar

Selecting the Form Full Screen selection again will set the edit mode to Standard Form Mode if the Dialog Box is small enough to fit in the Standard Form Mode Area. If not you must resize the Dialog Box to fit in this area before switching back to Standard Form Mode.

Font System

The EGUI Font System is based on the built-in EGA/VGA Programmable Character Generator. The EGA/VGA Video Sub-System allows a character set to be downloaded. This is taken care of by the System BIOS and the EGUI Initialization Procedure.

Important: On a EGA Video Adapter the 9x16 Bold Font *(EGUI System Font 0)* is not available. Only EGUI System Fonts 1 through 6 may be used in display mode 9 on this adapter.

Any font may be used with any Control Objects in the EGUI System that uses fonts. Below is a list of the available fonts in the EGUI System.

◆ EGUI System Font Numbers

0 = 8x16	**'Bold**
1 = 8x14	**'Bold**
2 = 8x14	'Normal
3 = 8x14	*'Italic*
4 = 8x8	**'Bold**
5 = 8x8	'Normal
6 = 8x8	*'Italic*

◆ EGUI System Font File (BWSYS.FNT)

The font file **BWSYS.FNT** contains the additional downloadable font sets used by the EGUI Font System which is loaded at initialization time.. This file must be distributed with any applications developed with the EGUI Library for proper operation and should be located in the same directory as the EGUI.INI file.

The following Library Function Procedures are used to manipulate the font system. See Chapter 13 for a definition and usage of these procedures.

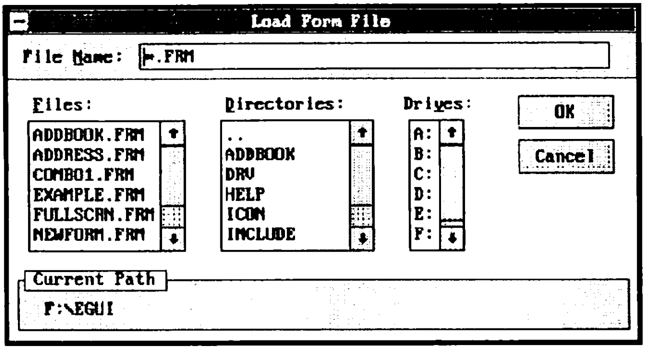
gSetEnvFontNum%
gGetSysFontHgt%
gLoadSysFont%

Also see definitions on the following Environment Font Properties in Chapter 10.

FontBackColr
FontForeColr
FontHgt
FontWid
FontTrans

The File Manager

The process of manipulating files in the EGUI System (i.e. creating, saving, loading, etc.) are all done with the File Manager.



Load Form File

File Name: *.FRM

Files:
ADDBOOK.FRM
ADDRESS.FRM
COMBO1.FRM
EXAMPLE.FRM
FULLSCRN.FRM
NEWFORM.FRM

Directories:
..
ADDBOOK
DRV
HELP
ICON
INCLUDE

Drives:
A:
B:
C:
D:
E:
F:

Current Path
F:\EGUI

EGUI File Manager

When the File Manager opens it will have the topic of the task it is performing in the Title Bar (example: **Load Form File**). Also the file specification will reflect the current task. Below is a brief definition of how to use the File Manager's Controls.

◆ File Name: Edit Box

Enter the file name of the Form you wish to manipulate. Be sure to add a **.FRM** extension if not already present. You may optionally enter a drive or directory path. After entering the file information press ENTER or click the OK button to select.

◆ Files: List Box

The *File List Box* contains a list of files from the current path which meets the current file specification. Highlight a selection from this box and press ENTER or click the OK button to select a file.

◆ Directories: List Box

The *Directories List Box* contains a list of the sub directories from the current path. Highlight a selection from this box and press ENTER or click the OK button to select that directory. After selection the path will be changed to the selected directory and a new file list and directory list will be shown.

◆ Drives: List Box

The Drives List Box contains a list of the drives which exists on the system. Highlight a selection from this box an press ENTER or click the OK button to select the drive. After selection the drive will be changed to the selected drive and the active directory for that drive, then a new file list and directory list will be shown.

◆ Current Path

This is a display box which shows the currently selected drive and directory path.

◆ OK Button

Click the *OK* button with the left mouse button to select a file, directory or drive entry.

◆ Cancel Button

Click the *Cancel* button with the left mouse button to exit the selection process.

◆ File Specification

The file search specification which appears in the *File Name Edit Box* when the File Manager is first opened (i.e. *.FRM). This may be reset to any legal DOS file search specification. (See you DOS Manual for more information.)

Drawing the Interface

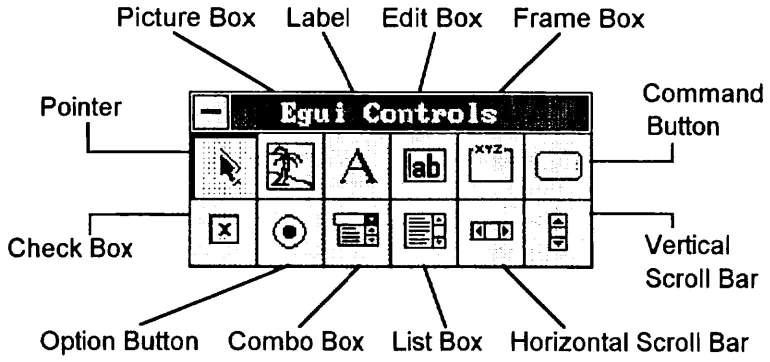
Chapter 4

Drawing Controls

You create the interface of your application by drawing Control Objects on a Dialog Box Form. A Control Object may be selected from the Control Menu or from the Control Object Tool Box.

◆ Control Object Tool Box

The Tool Box is the primary way of drawing Controls to a Form. The Tool Box is in a Dialog Box which may be moved around the display as to allow the viewing of information under the box. There are Icon Buttons on the Tool Box which represents each one of the Control Objects and a Pointer for allowing the manipulation of those controls.

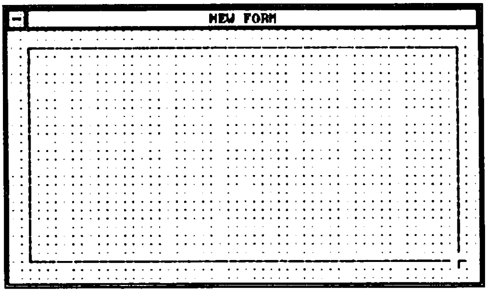


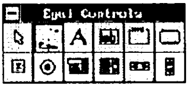
Control Object Tool Box

To display the Tool Box select **Control Tool Box** form the View Menu or click the right mouse button. When the Tool Box opens the Pointer Control is already selected. This is because any time the Form is selected you may manipulate any controls you wish. To select a Control to draw click the left mouse button on the desired icon. Once you release the button the Tool Box will close and a mouse crosshair will be placed in the center of the form. To close the Tool Box without selecting a control click the Close Button or click the right mouse button.

Drawing a Picture Box

Click the left mouse button on the Picture Box Control Icon and release the button. A mouse crosshair will appear in the center of the form. Move the crosshair to the position you wish to place the upper left corner of the Picture Box Control at and click the left mouse button and release it. Then drag the other corner of the rectangular box to the position of the lower left corner and click the left mouse button again this will set the Control Objects Boundaries.





Drawing a
Picture Box Control

To load a picture into the box select **Select Picture File** from the Tools Menu or press SHIFT-F8 and select the PCX image file you wish to load.

Note if the image is larger than the Picture Box the image will be clipped.

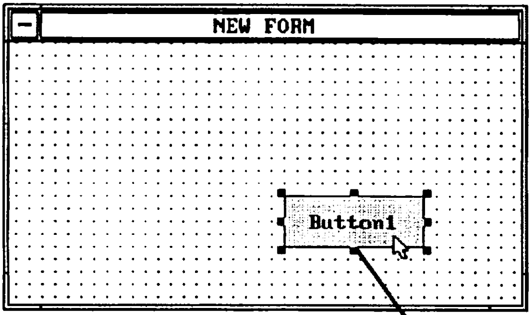
All other Control Objects are drawn to the Form in the same manner as the Picture Box.

Editing Controls

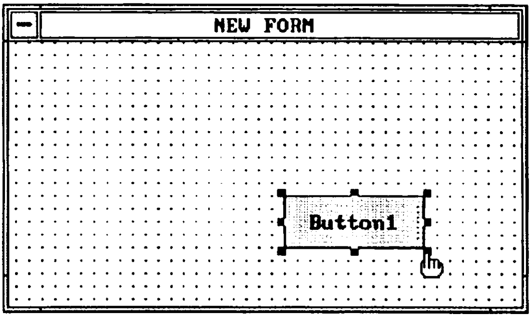
After you draw a control you can change its size, move it, or delete it.

◆ Resizing Controls

1) Click the control to select it. The control will be outlined with small rectangles called *sizing handles*, as shown below.



Click Control to display the Sizing Handles



2) To size the height and width of the control at the same time, drag one of the corner sizing handles, then release the mouse button to redraw the control.

   To size the control in one direction only, drag a sizing handle on one of the sides, then release the mouse button to redraw the control.

◆ Moving Controls

To move a control position the mouse pointer anywhere inside the control border and drag it to its new location.

Note: Controls located inside of a Frame Box will move with the Frame Box. In fact if you need to move more than one control at a time draw a Frame Box around the controls you wish to move, then move the frame, and then delete the frame.

◆ Deleting Controls

1) Click the control to select it.
2) Press the DEL key or select **Delete** from the Edit Menu.

◆ Copying Controls

1) Click the control to select it.
2) Press the CTRL-INS keys or select **Copy** from the Edit Menu.

Note: You may use the **Cut** command from the Edit Menu to copy and delete a control simultaneously. Press SHIFT-DEL to perform a cut process.

◆ Pasting Controls

1) Press the SHIFT-INS keys or select **Paste** from the Edit Menu.
2) The new control will be pasted in the upper left corner of the form. Then Drag the control to its new position.

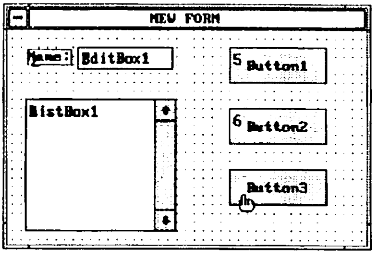
◆ Undo Edit

When you delete or cut a control a copy of that control is maintained in the Undo Buffer. Select **Undo** from the Edit Menu to undo your last delete or cut process.

Setting the Control Tab Order

The *Tab Order* is the order in which the focus is moved from each control when the TAB or SHIFT-TAB keys are pressed. Controls are placed in the Tab Order they are created in.

To reorder the Tab Order select **Control Tab Order** from the Edit Menu. The mouse cursor will change to a Hand, place the Hand Cursor inside the Control Border and click the left button. The Control Border will be highlighted and the new tab order number will be displayed inside the control border. After you have clicked every Control in the form the form will refresh in the new Tab Order. To stop the process click the right button.



Refreshing the Form

When drawing or editing controls on a form sometimes there is paint residue left over from other controls. To clean up this residue select **Refresh Form** from the Edit Menu and the form will be redrawn.

The short cut key **F4** may be used to refresh the form any time you are in the edit mode or have a control selected. This should be much more convenient then selecting this process from the pull down menu.

Go to Form Key

You may select **Go to Form** from the View Menu , or use the **F6** short cut key, to move to the last selected control on a Form from the Main Menu Bar. This is the same as clicking a control with the mouse.

Aligning Controls on the Form

The EGUI Form Generator has a built-in Grid system to help align controls on a form. When a control is moved you may have noticed that it snaps to a grid position on the form. This is because the controls are being aligned to the internal grid. The upper left corner of the control border is what gets aligned to the grid. **Note:** that other portions of some control borders may not align with the grid.

The Grid System is set on 8 Pixel Increments. This means that when the grid is on a dot will be displayed every 8 pixels, both horizontally and vertically. Inside the Dialog Box Form Border only.

Alignment to this grid is needed for some of the controls to operate property. The Controls which should be aligned to the grid system are:

Edit Box
List Box
Combo Box
Picture Box

All other Controls may be optionally aligned using the grid system.

Important: Moving or Resizing these Controls with the snap turned off may cause undesirable and unpredictable effects.

You may turn the Dotted Grid on and off by selecting **Grid** from the Edit Menu.

The snap works independent of the of the Grid System. The snap process is what actually aligns the Controls to the Grid. The snap may be turn on and off by selecting **Snap** from the Edit Menu.

This allows the Grid to be displayed and used as an alignment tool without the snap process occurring.

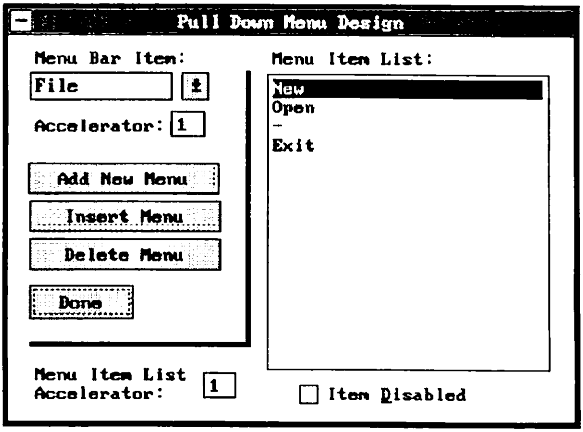
Pull Down Menu Design

To add a Pull Down Menu to a Form select **Pull Down Menu Design** from the View Menu. By default the first menu bar item is already created. You may rename it to anything you wish by selecting the Menu Bar Item Edit Box and typing the new Item Name.

Important: Make sure to press the ENTER key after an entry in this edit box to confirm your changes.

◆ Adding Items to a Menu List

By default place holders for 15 menu items are created for each menu bar item. This is the maximum number of items that can be added at Design Time, however this index may be increased in the code after generating source code for the form if needed. Select the Menu Item List to added and change your menu items. Highlight the item you wish to edit and type the changes and press the ENTER key to confirm your changes.



Menu Bar Item: File

Accelerator: 1

Menu Item List:
New
Open
-
Exit

Menu Item List Accelerator: 1

□ Item Disabled

◆ Disable a Menu List Item

To Disable and item in the menu list highlight that item in the list and click the Item Disabled Check Box.

Adding Accelerator Keys

To add an Accelerator key to a **Menu Bar Item** select the Accelerator Edit Box under the Menu Bar Item Edit Box and enter the character number to use as the Accelerator key.

To add an Accelerator key to a **Menu List Item** highlight the menu list item then select the Accelerator Edit Box under the Menu Item List Edit Box and enter the character number to use as the Accelerator key.

Adding a New Menu

Click the Add New Menu Button and a New Menu List will be added to the end of the current menu bar lists. A maximum of 8 menu lists may be added at design time. This may be increase in code.

Inserting a New Menu

To Insert a Menu Item List select the menu bar item which will follow the insertion and click the Insert Menu Button.

Deleting a Menu List

To Delete a Menu Item List select the menu bar item to delete and click the Delete Menu Button.

After making all your changes to the menu design click the Done Button.

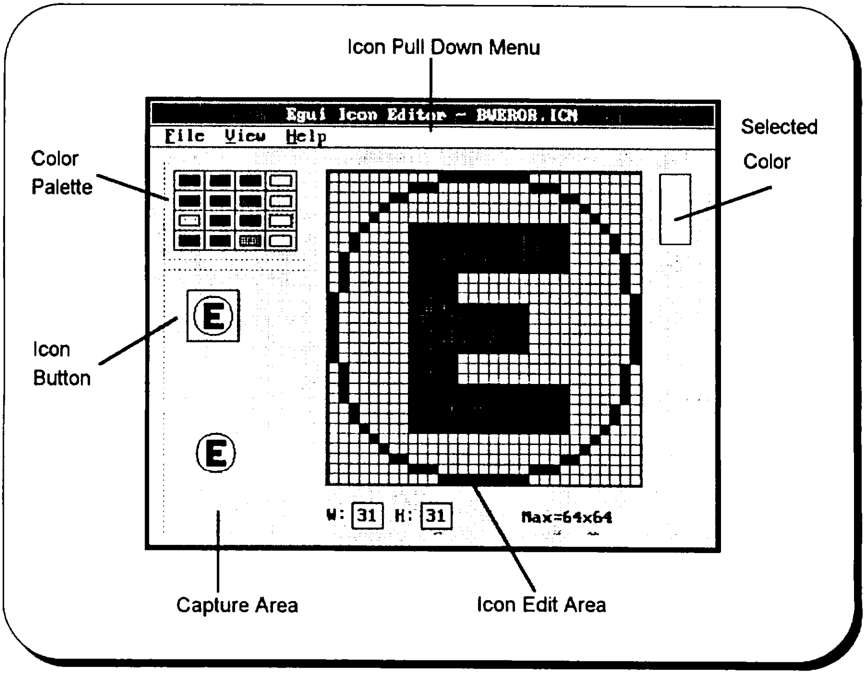
Important: To make the Pull Down Menu active you must select the Dialog Box Control by clicking on a part of the Form inside of the Forms Border but outside of all other controls borders. Then set the **pdmenuflag** property to **True**.

Icon Editor

Icons are graphical representations of an object or concept. They may represent objects that the user wants to work on or actions that the user wants to perform. The EGUI System comes with an Icon Editor which may be used to create Icons to be displayed on Command Buttons or on the Dialog Box Form itself. Icon Files in the EGUI System have a .ICN file extension. **Note:** Displaying Icons on a form must be done in code, it is not supported in the form generator.

◆ Loading the Icon Editor

Select **Icon Editor** form the View menu.



◆ Loading an Icon File

To load an existing Icon File select **Load** from the Icon Editor File Menu. Then select the icon file you wish to load from the file list

Creating an Icon File

Select New form the Icon Editor File Menu. The Icon Edit Area is cleared and set to a new icon size of 64 x 64. This size may be readjusted by selecting the width (W:) and height (H:) edit box at the bottom of the editing area and entering any value between 1 and 64.

Then select a color to paint the icon with from the Color Palette and click the mouse pointer inside the Icon Edit Area to paint the pixels of the icon.

The icon is painted in three places. The Icon Edit Area , Capture Area and the Icon Button. The Capture Area is used to hold a copy of what the icon will look like when captured. The Icon Button is used to give you and idea what the icon design is going to look like on a Command Button.

Saving an Icon File

After you have completed your icon select **Save As** from the Icon Editor File Menu. Enter a valid DOS file name with the extension .ICN and click the OK button. You may optionally select a new directory or drive also.

Icon Editing Grid

The Icon Editor has a built-in grid system that may be turn on and off from the View Menu by selecting **Grid**. This will help in locating each individual pixel.

Icon Edit Zooming

There are also **Zoom In** and **Zoom Out** features located in the View Menu which may be used to increase and decrease the Icon Edit Area size.

Important: You may not Zoom In on an Icon which is larger than 32 x 32 pixels.

Tools Menu

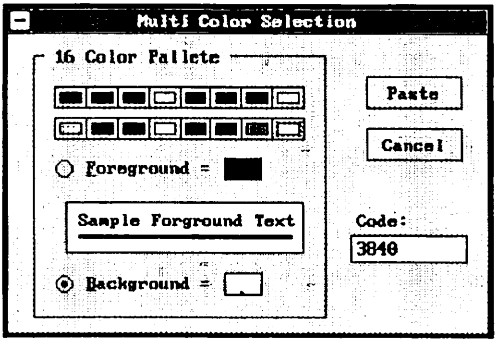
The Tools Menu has a list of tools to aid in the setting and manipulation of controls and properties.

◆ Calculate Multi Color Value

Some control properties require the assignment of a **Multi Color Value** *(see Chapter 3 Color System for more information on a Multi Color Value).* This tool will calculate this value for you by selecting the properties foreground and background colors.

Note: Before using this tool you must select a control which requires this value. If the selected control does not require this value the menu selection will be unselectable (grayed out).

To use this tool select **Calc Multi Color Value** from the Tools Menu or press **F9** if a control is selected.



Multi Color Selection

16 Color Pallete

○ Foreground =

Sample Forground Text

● Background =

Code: 3840

Multi Color Calculation Tool

By default the background option is selected when the window opens. To select the background color click a color on the color palette bars. The selected color will be displayed to the right of the Background option and the code for that color placed in the code box.

Then click the foreground option button to select foreground and pick a color from the palette bars.

The Multi Color Value will be calculated and displayed in the code box and a sample of your color selection is displayed in the sample box between the foreground and background option buttons..

Click the PASTE button to paste this value into the property or Cancel to stop the process.

Note: The effect of this value on the control depends on which control is selected. See the property definitions for the selected control to determine the correct selections.

◆ Icon File Selection

To select an Icon file for a control pick **Select Icon File** from the Tools Menu or press **SHIFT-F7** if a control is selected. The File Manager will open allowing you to select an Icon File *(see Chapter 3 The File Manager fro more information on using the File Manager).*

◆ Picture File Selection

To select a picture file (.PCX) for a control pick **Select Picture File** from the Tools Menu or press **SHIFT-F8** if a control is selected. The File Manager will open allowing you to select a Picture File *(see Chapter 3 The File Manager fro more information on using the File Manager).*

Setting Properties

Chapter 5

The Role of Properties

A property is a named attribute of a Control Object which may be set to define one of the characteristics of that object (such as size, color, font type, etc.) or an aspect of its behavior. This chapter explains how to set properties at Design Time using the Control Property Bar. Also how to set properties in code at start up time and during run time. Properties of Control Objects are used to fine tune that objects actions and appearance. This gives you a great deal of control over an Object.

Each Control Object in the EGUI System has a predefined set of properties which are set to a default value at creation time. For example a *List Box* control has a *Vertical Scroll Bar* active at creation time. If you do not need a scroll bar you may turn it off by setting the **novsbflag** property to True in the Properties Box on the Control Properties Bar.

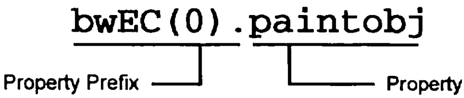
Note: You do not have to set every property, only the ones you wish to change the values of. See Chapter 11 Control Properties for definitions of each of the different control properties.

◆ Property Prefix

Most properties in the EGUI System must have a property prefix when assigning information to that property in code. A property prefix is actually the User Defined Type Identifier for the BASIC Language. Most properties in the system are defined in a user define type structure (Type-End Type).

Very Important: The EGUI System only declares one instance of each property type. The same memory location is used to pass property information to a control for each instance of that control. This will require that the Control's properties be set before every call to the control.

You will find the property prefix for each Control with the definition of that control in Chapter 12 under the sub section **Property Prefix**.

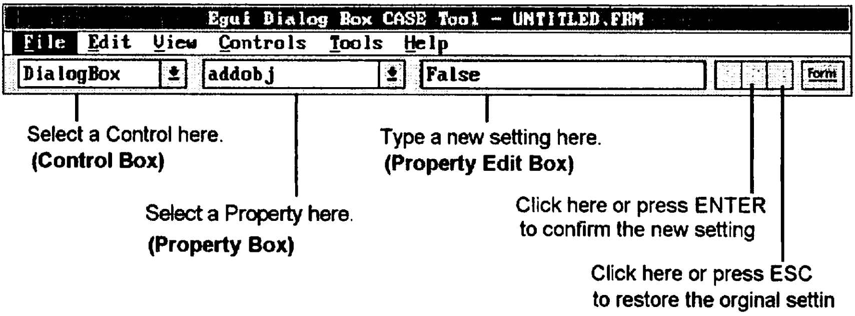


```
bwEC(0).paintobj
```

Using the Control Property Bar

The Control Properties Bar is locate below the Main Menu Bar as shown below. You use this bar to select and set properties for Control Objects.

Main Menu Bar & Control Property Bar


Egui Dialog Box CASE Tool - UNTITLED.FRM
File Edit View Controls Tools Help
DialogBox
addobj
False
Form

Select a Control here.
(Control Box)

Select a Property here.
(Property Box)

Type a new setting here.
(Property Edit Box)

Click here or press ENTER
to confirm the new setting

Click here or press ESC
to restore the orginal settin

◆ Editing a Control Property

1) Select the control by clicking it with the mouse pointer. The Control Name will appear in the **Control Box**. You may also click the Control Box Combo Button and then select a control from the list.

2) Click the **Property Box** Combo Button and select the property from the list that you wish to edit.

3) Click the **Property Edit Box** to enter a new value. **Note:** If the Property Edit Box Combo Button is active you may only pick an option from the Drop Down List. You may not type an entry in this case. To pick from the drop down list click the combo button to the right of the Property Edit Box. Select from the list and then press ENTER or click the confirm button.

Other Ways to set Properties

Properties are set at Design Time using the method describe in **Using the Control Property Bar**. Property may also be set in code at Run Time. The example code below shows how to set properties at Run Time for a Horizontal Scroll Bar. Use this example to help understand setting properties at Run Time. Also see Chapter 9 Attaching Code to Controls for more information. **Note:** Remember you must create source code and load it into the IDE System or compile it for Run Time Mode.

```
'This code should be located in the Draw Dialog Box Section
'of the DBFormat
'Setup Horz Scroll Bar Number 1 ----------------------------

 GOSUB EXAMPLEReSetHorzSBar1
 bwSB(0).paintobj = -1
 bwSB(0).addobj = -1
 bwSB(0).update = -1
 retcode% = gHorzScrollBar%

'Place this code in a Case Statement in the Window Main Loop
 CASE case_select_number

     GOSUB EXAMPLEReSetHorzSBar1
     retcode% = gHorzScrollBar%
     bwHorzSBarVal1! = bwSB(0).value
     IF retcode% = -3 THEN
         GOSUB EXAMPLEProcessKey
     END IF

'Place this code in the Dialog Box Properties Section of the
'DBFormat
 EXAMPLEReSetHorzSBar1:
     bwSB(0).paintobj = 0
     bwSB(0).addobj = 0
     bwSB(0).update = 0
     bwSB(0).tabflag = -1
     bwSB(0).enable = -1
     bwSB(0).objid = -1
     bwSB(0).status = 0
     bwSB(0).x1 = (x1% + 24)
     bwSB(0).y1 = (y1% + 288)
     bwSB(0).objheight = bwStandButwid%
     bwSB(0).objwidth = 400
     bwSB(0).frame = 20
     bwSB(0).value = bwHorzSBarVal1!
     bwSB(0).min = 0
     bwSB(0).max = 100
     bwSB(0).smallchange = 2
     bwSB(0).largechange = 20
 RETURN
```

System Environment Properties

The EGUI System has a set of predefined System Environment Properties which are used to control the overall application appearance and behavior. These are attributes like Title Bar Color, Border Width, Default System Font Number, Dialog Box Color and more.

These properties are set in the same manner as Control Properties are in code. There is no method of setting these properties in the EGUI Form Generator, however they may be set in the **EGUI.INI** file.

The default settings, which may be adjust in the EGUI.INI file (see Appendix F for information on the EGUI Initialization File Format), are loaded at **System Initialization Time**.

Important: Once you have set a System Environment Property this property value may be used throughout the application and shared by different Control Objects. Be sure to save old settings if they will need to be reset.

Note: Some Environment Properties have duplicate Control Properties. Example **bwEV(0).DialogBoxcolr** and **bwDB(0).dcolr** are both used to set the Dialog Box Control background color. In the cases where there are duplicate properties the Control Property will override the System Environment Property if it is set to a valid value. Usually if the Control Property is set to zero (0) the Environment Property is used. See the control property definitions in Chapter 11 for more information.

See **Chapter 10 Environment Properties** for a complete definition on all the available System Environment Properties.

Creating Form Source Code

Chapter 6

The Source Code Compiler

The EGUI System use a Source Code Generator (Compiler) Application to interpret the Form Files and build BASIC Source Code Files. The Source Code Files may then be loaded into the IDE System or Compiled by the BASIC Compiler.

The Source Code Files (**.FNC** or **.BAS** files) may be built from the DOS Command Line or from inside the EGUI Form Generator. This section covers the use of the Source Code Compiler from the DOS Command Line. See the next section in this chapter *Building Source Code Modules* for information on how to use the Source Code Compiler from inside the Form Generator.

Below is the syntax for the Source Code Compiler. To get a help screen from the command line type EGUISGEN /H.

◆ Source Code Compiler Syntax

EGUISGEN [*path*]*filename*.FRM /FN=*functionname* [/S] [/L] [/BM] [/SC]

◆ Source Code Compiler Switch Definitions

/FN= Sets the Function Name. This is NOT Optional. The Function Name is the name of the procedure you are going to build. This may be any valid BASIC Function Procedure name.

Important: Because this name is concatenated with other local procedure names you must keep the name length to a maximum of 20 or less (the fewer the better).

/BM Sets the Build Module Flag. This flag will cause the Source Code Compiler to build a BASIC Module with the EGUI Include File Code already in it.

If this switch is not set the Compiler will build a Function File which must be merged into your module code. This can be done from inside of the IDE System by first loading the Module Level Code an then selecting Merge from the File Menu and merging this function into the module.

/SC Sets the Build Module with StartUp Code Flag, the **/BM** switch **must be set** prior to this switch.

The EGUI System requires some startup code to Initialize the System. If you are building a new module it is recommended to use this switch. After this module is built it may be loaded into the IDE System (or Compiled) just like any other BASIC Modules. This should be you Main Module Code.

/S Sets the STATIC Flag. This will cause the Compiler to place the **STATIC** keyword after the Function Name. This switch is not recommend unless you must maintain all the variable information in the procedure during calls. This is usually not needed.

/L Sets the **LOCAL ERROR** Flag. The Compiler will build Local Error Checking Code into your procedure if this switch is set.

Important: This switch is only compatible with MS PDS 7.x and VBDOS 1.0. Do not use this switch with procedures being built for the QB 4.x Compiler.

The Source Code Compiler will name the New Source Code File the same name as the Form File with the new extension, either **.BAS** or **.FNC** which every is appropriate.

Note: If the BASIC Module or Function File already exists the Compiler will prompt you for the go ahead to rebuild the procedure before building it.

Building Source Code Modules

You may build Source Code Files from within the EGUI Form Generator. If you elect to, the Generator will shell out and invoke the Source Code Compiler the reload the Generator and current form.

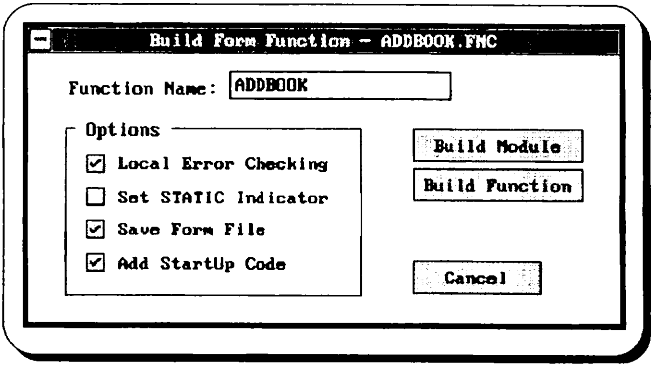
◆ Building a Source Code File

1) Select **Create Form Function** from the File Menu. The Build Form Function Dialog Box will open.

2) Enter a Function Name in the Edit Box. By default the form name will appear in the Edit Box this may be changed to any valid BASIC Function Name (a Maximum of 20 Characters).

3) Set any options from the Option Frame Box you desire.

4) Click the Build Module button if you wish to build a BASIC Module or the Build Function button if you want a Function File.



Build Form Function - ADDBOOK.FNC

Function Name: ADDBOOK

Options

☑ Local Error Checking

☐ Set STATIC Indicator

☑ Save Form File

☑ Add StartUp Code

The Source Code Compiler will be invoked and the Source Code File will be placed in the same location as the Form File.

If you build a Function File this code must be merged into a BASIC Module. This can be done from inside of the IDE System by first loading the Module Level Code an then selecting Merge from the File Menu and merging this function into the module.

Printing Forms

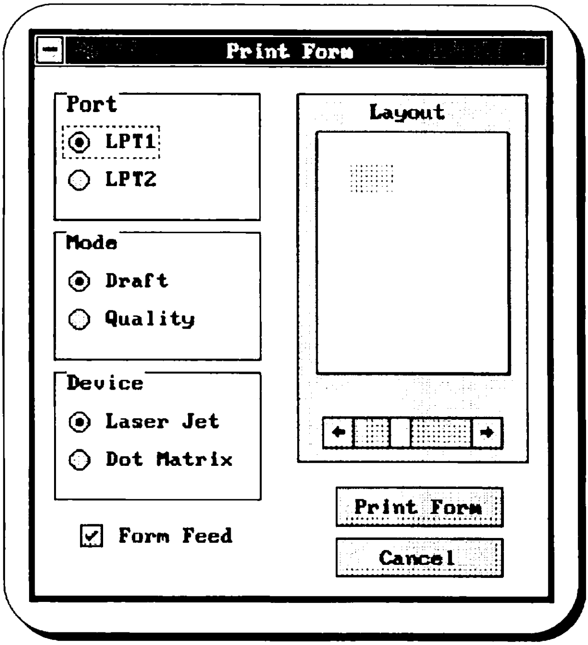
Chapter 7

Printing a Form Image

You may print Form Images to a HP Laser Printer or an IBM Dot Matrix Printer or 100% compatible printer. The best results will be accomplished with a laser printer.

Select **Print** from the File Menu and the Print Form Dialog Box will open. The Dark Gray rectangle on the Layout Sheet is a representation of your form. By default the form will be set to its actual size when you open the Dialog Box. You may scale the form print size up or down by clicking the Horizontal Scroll Bar. The form may be printed in portrait mode only.

After selecting any option necessary click the Print Form button to print.



Print Form

Port
● LPT1
○ LPT2

Mode
● Draft
○ Quality

Device
● Laser Jet
○ Dot Matrix

☑ Form Feed

A Dialog Box will open showing the percentage done of the print process. Once the process is complete both Dialog Boxes will close.

EGUI Toolkit
Part 3

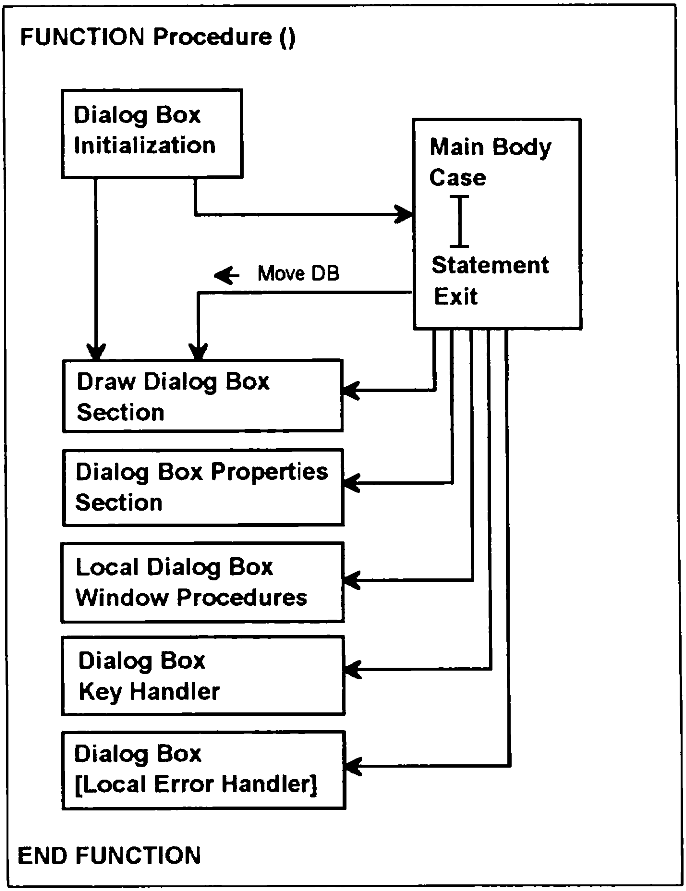
Application Development

Structuring the Application

Chapter 8

DBFormat Structure *(Dialog Box Code Format)*

The **EGUI DBFormat** is simply a way of organizing your code in a function procedure so that it is much easier to develop. Below is a block diagram of the **DBFormat** Structure. **Important:** Using the DBFormat is optional, because we understand that being the veteran programmer that you are, you may already have a procedure format you are comfortable with. The EGUI System will work find in many formats, however we use this format to indicate where to find code and where to put code when you are attaching code to your application. Also the EGUI Source Code Compiler uses this format when building dialog box code. So it is recommended that you are at least familiar with this format.


FUNCTION Procedure ()
Dialog Box Initialization
Main Body
Case
Statement
Exit
Move DB
Draw Dialog Box Section
Dialog Box Properties Section
Local Dialog Box Window Procedures
Dialog Box Key Handler
Dialog Box [Local Error Handler]
END FUNCTION

See the source code in Chapter 12 under Pull Down Menu Control to help associate yourself with the format in code.

Important: Make sure any time you exit the dialog box procedure you exit through the local exit point. This will make sure that the Dialog Box will close properly and remove all its objects from the object list.

Loading Form Source Code

After you have produced some form source code with the EGUI Source Code Compiler you may load it into the BASIC IDE System or into your favorite Text Editor.

If you are using the IDE system and are loading a **Source Code Module** which should have an extension of .BAS the select **Open** or **Load** from the **File** menu.

If you are using the IDE system and are loading a **Source Code Function File** which should have an extension of .FNC the place your cursor at the insertion point in your code and select **Merge** from the **File** menu. The procedure should be loaded and you can move to the procedure by pressing **F2** and selecting it from the SUBs Window .If your are loading a function file into a text editor it is recommended to merge the code at the end of you current code.

Initializing the EGUI System

It is necessary to initialize the EGUI System before using it. If you are creating you main module code from the Form Generator this step is taken care of for you. If not the following code should go in your main module before you call any procedures in the EGUI System.

```
'Clear Extra Stack Space -------------------------------------
 CLEAR , , 3000

'Set Standard Screen Mode 12 VGA (640x$80 16 Color ) ---------
 Vmode% = 12        '9 for EGA
 ClrScrflag% = -1
 retcode% = gSetVideoMode%(Vmode%, ClrScrflag%)

'Initialize Egui System --------------------------------------
 IF NOT retcode% THEN
     retcode% = gInitBWSystem%
 ELSE
     GOTO MainExit          'This will exit if an error occured
END IF
```

Display Modes & Viewports

The EGUI System with the Standard Video Driver Module only supports EGA, BASIC's screen mode 9, and VGA BASIC's screen mode 12. VGA screen mode 12 is the recommend mode to use for most of your application. These two modes are the most optimal graphics screen modes that BASIC supports. Meaning that their are a lot of applications which may be developed using these modes without any lose in features.

◆ Supported Display Modes for the SVDM

EGA - BASIC's Screen Mode 9
VGA - BASIC's Screen Mode 12
TEXT - BASIC's Screen Mode 0 (this mode may be set on exit)

These modes should be set in the main module before any other library routines are called. Use **gSetVideoMode%** to set the video mode *(see Chapter 13 for more information on this procedure).* Note you may also reset to Text Mode, screen mode 0, just before ending you program.

When creating a BASIC Source Code Module from the EGUI Generator the default screen mode is VGA. All the code needed to set the proper screen mode will be built for you.

Important: The **SVDM** uses most of the BASIC's built in Graphics Primitives (i.e. Line, Circle, etc.), with a few addition that BASIC left out. Be sure to read Chapter 13 and note the similarities and differences between these primitives.

The Standard Video Driver Module **(SVDM)** is the default Video Driver Module which comes with the EGUI System. There will be an Enhanced Video Driver Module **(EVDM)** available as an add on product which will support other Graphics Libraries. This driver, in combination with the other graphics libraries, will allow you to create application in about 15 to 20 additional screen modes, from Hercules 720x350 2 color to Super VGA 1024x768 256 color, and many modes in between. BASIC itself by its self does not support these additional modes. Note the EVDM will impose different hardware requirements, which will be specify with the driver.

◆ Viewports

The only viewport that the EGUI System use is the full screen viewport. The BASIC **View** or **Window** statements are not used to do any clipping of the graphics primitive elements of the system. In fact there is very little clipping done in the library at all. This is because clipping adds a great deal of overhead in processing time of certain procedures.

Be careful when using these statements in your code. This is because the SVDM uses BASIC's built in Graphics Primitives, and when you use these commands you also effect the SVDM. Also remember that the addition primitives that the EGUI System has added do not respond to these statements in the SVDM.

Important: There are some properties which exist in the EGUI System which will not have and effect on the **SVDM**. They are intended for use with the **EVDM**.

Window Types (Modal Dialog Boxes)

The EGUI System only supports Modal type Windows or Dialog Boxes.

◆ Modal Dialog Box

A **Modal** Form is a Dialog Box which once open retains the focus until the window is closed. You may not move the focus to another Dialog Box until the current Dialog Box has been closed.

This is similar to many existing DOS application today and should cause no problems for application development.

A Dialog Box may have a wide border or a thin border. This is selected with the Dialog Box Control's **nonborderflag** property. The purpose of the Dialog Box border in the current EGUI System is for aesthetics. However in future versions we hope to implement the ability to size a Dialog Box. Then the border types will play a role in this process. You might keep this in mind when developing your applications. A wide border will be used to indicate that it is sizable and a thin border will indicate that it is not sizable.

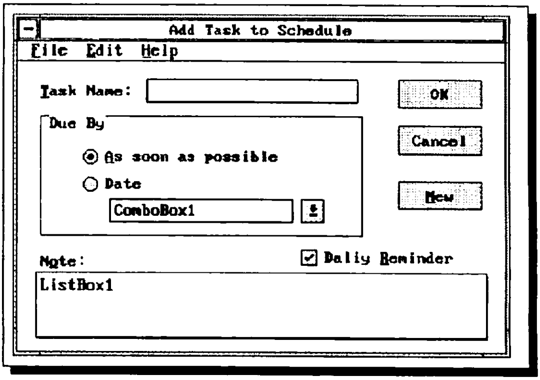
◆ Dialog Box Instances

There may be a maximum of 7 Dialog Boxes open at one time. Each one of these windows are consider an instance of the Dialog Box Control.

The first instance of a Dialog Box Control functions a little differently from all other instances. The first instance will allow you to design controls and place them outside of the Dialog Box Border. Then if a mouse event occurs outside the dialog box border the system will react to the event. This will allow you more freedom when creating your main Dialog Box Control.

Instances two through seven of the dialog box control will not react to any controls placed outside the border of there parent Dialog Box. In fact if you click a mouse outside this area you will get a beep and no other processing will continue until you release the mouse button.

Important: You should never place a control outside of any Dialog Box Control border except for the first instance of a Dialog Box. But remember there is no clipping done in the EGUI System so it is up to you to may sure that a control's border remain inside its parent the Dialog Boxes' border. This process is done for you in the EGUI Generator, however once the source code is created the balls in your court.



● As soon as possible

○ Date

ComboBox1

☑ Daliy Reminder

ListBox1

Where and How to Use Control Objects

The overall topic of User Interface Design and how an applications focus is organized is an extremely large subject and is beyond the scope of this manual. However we would like to cover some of the more basic User Interface Design techniques and give you some tips for controlling the focus of your application development.

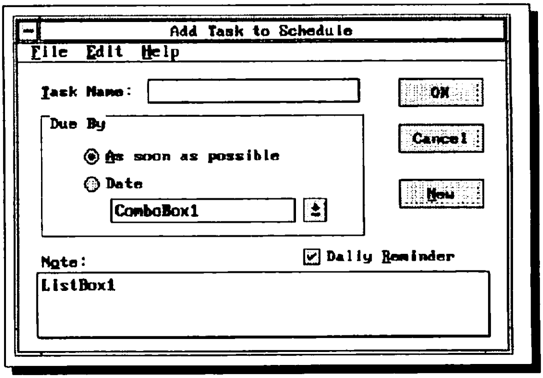
The EGUI System uses *Control Objects* to organize the focus in an application by giving the user a transfer point for communications. This transfer point allows the user to focus on the current topic and to interact with the application.

Below is a list of the supported controls in the EGUI System, and a description of there usage.

◆ Dialog Box Control gBuildDialogBox% & gRemoveDialogBox%

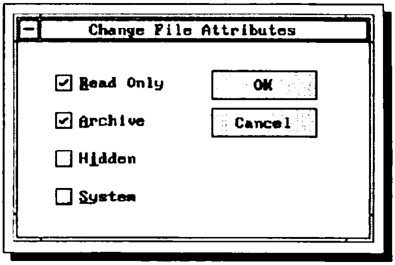
The Dialog Box Control is used to outline a portion of the display so that the user will focus only on that portion. The Dialog Box will contain user interface components that are supported by the EGUI System through controls that allow the user to select choices and enter information.

Below is a typical Dialog Box:



◆ Check Box Control gChkOptBox%

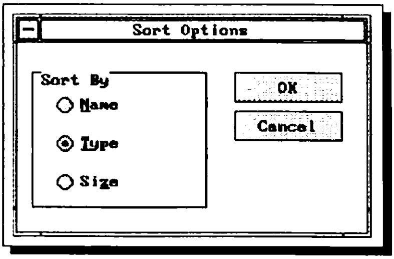
Check boxes control individual choices that are either turned on or off. When the choice is turned on, the check box shows a check mark, or a dark square or an X in it, depending on the setting of the **buttontype** property. When the choice is turned off, the check box is blank. The user can toggle the state of a check box by clicking on the box or by setting the focus on the check box and pressing the select key (SPACE-BAR). Below are some typical check boxes.



☑ Read Only
☑ Archive
□ Hidden
□ System

◆ Option Button Control gChkOptBox%

An option button represents a single choice in a limited set of mutually exclusive choices. Accordingly, in any group of option buttons, the user can only select one at any time. Option buttons are represented by circles which have the appearance of raised round buttons. When an option button choice is selected, the circle is filled; when the choice is not selected to circle is empty. If the number of option buttons in a group exceeds five it is recommended to replace the buttons with a dropdown list box. **Note:** The Form Generator produces the code necessary to make option buttons function in a group. If you are adding option buttons in code you must add this code yourself. Below are some typical option buttons.



Sort By
○ Name
● Type
○ Size

◆ Combo Box Control gComboBox%

A Combo Box is an Edit Box with an attached List Box. There are two types of comb boxes, a Standard Combo Box and a Dropdown List Box. The Standard Combo Box allows the user to enter text in the edit box or to pick an item from the list. A Dropdown List box will only allow the user to pick a choice from the drop down list.

When a combo box has the focus use the DOWN ARROW key or click the Combo Box Button to drop the list down. Once in the list the list will functions like a normal List Box Control. To make a selection highlight the item you wish to select and press ENTER or click inside the edit box control. The list will close and the focus is returned to the edit box.

You should use a Combo Box when the user may type a selection or pick it from a list.

See the Combo Box Control definition in Chapter 12 for an example.

◆ Command Button gCommandButton%

A Command Button is a graphical control that initiate an action when it is clicked or selected with the select key. The action taken is related to the label on the button. Example if you have a button that has the label Cancel on it, when this button is clicked it should cancel the current process and close the dialog box .

You should use an icon (picture) on a command button when the label may not be concisely represented with a textual label. You may display an icon on a button by setting the button **icon** property to True (-1) and set the icon file name in the **iconfile** property. You may also adjust the horizontal and vertical alignment of the icon with the **iconx1off** and **icony1off** properties.

You either display text or an icon on a command button you may not do both. And exception to this is if your icon has text in it.

See Chapter 12 for an example of a Command Button .

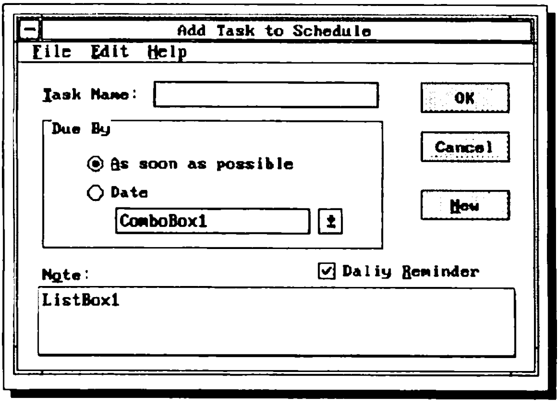
◆ Edit Box Control gEditBox%

Edit Boxes are controls into which the user types information. The user may except the current text , edit it or delete it. The LEFT and RIGHT ARROW keys are use to position the insertion point. The HOME and END keys are used to position the cursor at the beginning and ending of the text. The BACKSPACE key will erase the previous character and position the cursor to the left one character. The DELETE key will erase the current character, and the INSERT key will toggle the editing modes between *Overtype* and *Insert Modes.*

Note: When in Insert Mode you may only insert text until the edit box string length is at it's maximum length. One the maximum length is reached you must erase text before you may insert any more.

For a Multi-Line Edit Box use the List Box Control with the **editflag** property set to True (-1). This will combine the functionality of the List Box for scrolling and the Edit Box for editing. **Important:** There is no word-wrap capability in the multi line edit box. It functions only like a text editor. There are two line edit key features available, CTRL-L to insert a Line and CTRL-D to delete a line. **Note:** Once you have inserted text to the end of your list box array the you may not insert any more until you delete a line.

Below are some typical edit box and multi-line edit boxes.



◆ Scroll Bars gHorzScrollBar% gVertScrollBar%

Scroll Bars are graphical tools for quickly navigating through a long list of items or a large amount of information, and for indicating the current position on a scale.

Note: The Horizontal Scroll Bar in the EGUI System has a fixed height and the Vertical Scroll Bar has a fixed width.

The Scroll Box in the EGUI System will automatically adjust itself to represent the currently viewed portion of the unit the scroll bar is showing a measure for. The dark area before or after the scroll bar represents the portion of the unit that is not being viewed.

To move a scroll bar by its small change value click the Top/Left or Button/Right scroll buttons or when a scroll bar has the focus you may also use the UP and DOWN ARROW keys.

To move a scroll bar by its large change value click the dark areas of the scroll bar below or above the Scroll Box or when a scroll bar has the focus you may also use the PAGE-UP and PAGE-DOWN ARROW keys.

See Chapter 12 for an example of Scroll Bars.

◆ List Box gListBox%

The List Box Control displays a list of items from which the user may choose one. **Exception:** If the **tagflag** property is set to True (-1) you may pick multiple items.

A Vertical Scroll Bar is automatically displayed with a list box. This may be disable using the **novsbflag** property. You may also add a Horizontal Scroll Bar if needed by using the **hsbflag** property.

You may also add a Header and Separation lines to a list box. See the definition in Chapter 12 of a List Box for more information.

Attaching Code to Controls

Chapter 9

Event Procedures & Control Return Codes

Once you have created a Dialog Box Form and generated source code it will be necessary to attach code to the appropriate event to make your form fully functional. Each Control Object in the EGUI System returns a code which indicates what type of event was trapped. There are two primary events which may return an event code, see the listing below for a list of possible return codes.

	Event Return Codes	
Control	**Mouse Click**	**Key Press** +ALT or CTRL or Shift
Check Box		-3
Option Button		-3
Combo Box		-3
Command Button	-2	-3
Edit Box		-3
Scroll Bar		-3
List Box		-3
Pull Down Menu		-3

◆ Key Press Events

When a key press return code is set the actual key or key combination may be retrieved from the **KeyPress%** property. If a single key is pressed the key press property will equal the ASCII value of that key. If a shift key combination is pressed the key property will store that specific shift key combination. The values of these key combinations may be found in the appendix section of your BASIC manuals. Below are some example values.

Character	ASCII	Shift-	Ctrl-	Alt-
S	115	83	19	-31
F	102	70	6	-33
P	112	80	16	-25

Important: The Alt key combination will return a negative value of the key code. Also once the alt key has been depressed it will only trap the first occurrence of a key press, then the Alt key must be released before more key presses can be trapped.

◆ Mouse Click Events

The only control which returns a mouse click event directly is the Command Button. This is done when the mouse button has been fully depressed and released over the button. **Note:** the left mouse button is the button which is being trapped.

Note: It is recommended that you place code to test for an event after each call to a control. This process is done for you if you are using the form generator.

Key Press Event Handler

It is necessary to set up a section of code to respond to key press events. This process is none for you by the form generator, however you may need to add additional traps.

The purpose of the Key Press Event Handler is to give you a centralized location for responding to key presses.

It is recommended that the Key Press Handler be local to the procedure. This should reduce the number of variables which would have to be passed or made global so that and external routine could access them. Below is an example of a Key Press Event Handler.

```
'---------------------------------------------------------------
FUNCNAMEProcessKey:

    IF KeyPress% = -23 THEN                      'Key = ALT-I

    ELSEIF KeyPress% = -45 THEN                  'Key = ALT-X

    ELSEIF KeyPress% = 27 THEN                   'Key = Esc

        FUNCNAME% = 27                           'Cancle Process
        GOTO FUNCNAMEExit

    END IF

RETURN
```

Note: The key traps that are created by the form generator are the Controls and Labels which have an Accelerator Key assignment.

Status Property

Each Control Object in the EGUI System has a **status** property. If this property is set to True (-1) then that control will echo the event information that the control object itself is trapping. This will give you more flexibility for trapping events.

Very Important: Because this information is being returned back to your application this means you may do what ever you wish with that information at the time of the event. But remember that the event is being echoed back and the control object itself may not be completed with all of its processing. For this reason it is recommended that you store the event information, or only do short process and then pass control back to the control object as soon as possible.

> *Example:* If a Command Button is clicked the button will normally depress and then restore itself before passing the event back to your application. If the status property is set to true the event may be passed back before the button can restore itself.

The status property should be very useful but be careful not to interrupt the process of a control object or unpredictable results may happen.

Maintaining Control Pointers

There are four properties which maintain pointers to information about control objects **aptr, dptr, eptr** and **rptr**. These properties are set to default settings at startup time however they are changed at run time. You must set up temporary variables for storing this information. The EGUI Form Generator takes care of this for you , however if you add a control which uses one of these pointers you must add this code also.

The steps for adding this code are *(List Box is used for the example)*:

1) Setup temporary variables in the Dialog Box Initialization Section and assign the default startup values (see each property for their default values).

```
dptr1%=1
aptr1%=1
rptr1%=1
```

2) Use the temporary variable to assign the property value in the Dialog Box Properties Section.

```
bwLB(0).dptr=dptr1%
bwLB(0).aptr=aptr1%
bwLB(0).rptr=rptr1%
```

3) Update the temporary variable after returning from the control object in the DB Main Body .

```
CASE 10

     GOSUB EXAMPLEReSetListBox1
     retcode% = gListBox%(Array1$(), ArrayHeader1$)


     dptr1% = bwLB(0).dptr      'Update Index Pointers
     aptr1% = bwLB(0).aptr
     rptr1% = bwLB(0).rptr


     IF retcode% = -3 THEN
          GOSUB EXAMPLEProcessKey
     END IF
```

See the code generated by the Source Code Compiler to get more examples on how to build this code.

EGUI Toolkit
Part 4

Controls, Properties & Functions

Environment Properties

Chapter 10

EGUI System Environment *Properties*

Below is a list of description of the EGUI Environment Properties. These properties which are global to the system and are declared in the **BWENV.INC** include file, are used to control different aspects of the EGUI Environment as a whole.

Important: All environment properties must be prefixed with **bwEV(0).** for proper operation.

Property **Description**

ActiveCtrlNum

For control object's internal use. See Building Custom Controls *(Appendix B)* for more information.

Data Type: Integer

Black

Attribute place holder for the color black. See Color System for more information.

Data Type: Long

Blue

Attribute place holder for the color blue. See Color System for more information.

Data Type: Long

Bordercolr

The color to paint the active Dialog Box Border. The default color is light blue.

```
bwEV(0).Bordercolr = bwEV(0).LightBlue
```

Data Type: Long

Borderwid

The width to draw a Dialog Box Border. The default width is 3. Minimum = 2, Maximum = 8

```
bwEV(0).Borderwid = 3
```

Data Type: Integer

Brown

Attribute place holder for the color brown. See Color system for more information.

Data Type: Long

ChkBoxcolr

The color to paint the inside box of a Check Box Button when it is set active. By default this color is black.

```
bwEV(0).ChkBoxcolr = bwEV(0).Black
```

Data Type: Long

Controlcolr

The color to paint the background of an edit box or list box control. The default color is white.

```
bwEV(0).Controlcolr = bwEV(0).White
```

Data Type: Long

Controltextcolr

The color to paint the text in an edit box or list box control. The default color is black.

```
bwEV(0).Controltextcolr = bwEV(0).Black
```

Data Type: Long

CurFontNum

The current selected font type.

0 = 8x16	**'Bold**
1 = 8x14	**'Bold**
2 = 8x14	'Normal
3 = 8x14	*'Italic*
4 = 8x8	**'Bold**
5 = 8x8	'Normal
6 = 8x8	*'Italic*

Data Type: Integer

Cyan

Attribute place holder for the color cyan. See Color System for more information.

Data Type: Long

DarkGray

Attribute place holder for the color dark gray. See Color System for more information.

Data Type: Long

DefFontNum

The default font type, this may be set in the **EGUI.INI** file with **DefFontNum** =*fontnumber.*

0 = 8x16	**'Bold**
1 = 8x14	**'Bold**
2 = 8x14	'Normal
3 = 8x14	*'Italic*
4 = 8x8	**'Bold**
5 = 8x8	'Normal
6 = 8x8	*'Italic*

Data Type: Integer

DeskTopcolr

The color to paint the Desk Top Background during start up. The default color is light blue.

```
bwEV(0).DeskTopcolr = bwEV(0).LightBlue
```

This may be set in the **EGUI.INI** file with **DeskTopcolr=12** *(where 12 is the a color number 0-15).*

Data Type: Long

DialogBoxcolr

The color to paint a Dialog Box background. The default color is white.

```
bwEV(0).DialogBoxcolr = bwEV(0).White
```

This may be set in the **EGUI.INI** file with **DialogBoxcolr=15** *(where 15 is the a color number 0-15).*

Data Type: Long

DialogBoxtextcolr

The color to paint Dialog Box text. The default color is black.

```
bwEV(0).DialogBoxtextcolr = bwEV(0).Black
```

This may be set in the **EGUI.INI** file with **DialogBoxtextcolr=0** *(where 0 is the a color number 0-15).*

Data Type: Long

DisplayMode

The current screen display mode. This must be set to a value of 9 for EGA or 12 for VGA. See Display Modes and Viewports for more information.

Data Type: Integer

DoubleClickTime

The delay time that the system will wait for a second mouse click on the same object. The default is 6.

Minimum = 0
Maximum = 32

```
bwEV(0).DoubleClickTime = 6
```

This may be set in the **EGUI.INI** file with **DoubleClickTime=6**.

Data Type: Integer

FontBackcolr

The color to paint the active font background. The default is the value that the property **bwEV(0).DialogBoxcolr** is set to.

```
bwEV(0).FontBackcolr = bwEV(0).White
```

Data Type: Long

FontForecolr

The color to paint the active font foreground. The default is the value that the property **bwEV(0).DialogBoxtextcolr** is set to.

```
bwEV(0).FontBacktextcolr = bwEV(0).Black
```

Data Type: Long

FontHgt

This is the active font height. The value may be read by a program, but it is set buy the system and should **NOT** be reset.

Data Type: Integer

FontTrans

Flag which indicates that fonts will be drawn transparently if set to True.

```
bwEV(0).FontTrans = True    'or (-1)
```

Data Type: Integer

FontWid

This is the active font average width. The value may be read by a program, but it is set buy the system and should NOT be reset.

Data Type: Integer

FontUSOffset

For future development currently not supported.

Data Type: Integer

Gray

Attribute place holder for the color gray. See Color System for more information.

Data Type: Long

Green

Attribute place holder for the color green. See Color System for more information.

Data Type: Long

HighLightcolr

The color to highlite and item in a list box or combo box. The default value is white on black.

```
bwEV(0).HightLightcolr = _
          (bwEV(0).White   +   (bwEV(0).Black   *
256))
```

NOTE: This property uses the multi color formula.

Data Type: Long

InActBordercolr

The color to paint an inactive dialog box border. The default value is gray.

```
bwEV(0).InActBordercolr = bwEV(0).Gray
```

This property may be set in the **EGUI.INI** file with **InActBordercolr=7** *(where 7 is the a color number 0-15).*

Data Type: Long

InActTitlebarcolr

The color to paint an inactive dialog box title bar background. The default value is gray.

bwEV(0).InActTitlebarcolr = bwEV(0).Gray

This property may be set in the **EGUI.INI** file with **InActTitlebarcolr=8** *(where 8 is the a color number 0-15).*

Data Type: Long

InActTitlebartextcolr

The color to paint inactive dialog box title bar text. The default value is black.

bwEV(0).InActTitlebarcolr = bwEV(0).Black

This property may be set in the **EGUI.INI** file with **InActTitlebartextcolr=0** *(where 0 is the a color number 0-15).*

Data Type: Long

LightBlue

Attribute place holder for the color light blue. See Color System for more information.

Data Type: Long

LightCyan

Attribute place holder for the color light cyan. See Color System for more information.

Data Type: Long

LightGreen

Attribute place holder for the color light green. See Color System for more information.

Data Type: Long

LightMagenta

Attribute place holder for the color light magenta. See Color System for more information.

Data Type: Long

Linestyle

A 16-bit hex value integer mask used to draw pixel to the display by the **line** function. See **gDrawLine%** function for more information.

```
bwEV(0).Linestyle = &HFFFF    (Solid Line)
```

Data Type: Integer

Logicoper

An integer value indicating that a logical operator is to be used with a graphics function.

```
bwEV(0).Logicoper = 0    (PSET)
```

0 = PSET
1 = OR
2 = AND
3 = XOR

These parameters are different for displaying images and icons. See **gLoadIcon%** function for more information.

Data Type: Integer

Magenta

Attribute place holder for the color magenta. See Color System for more information.

Data Type: Long

Mouseflag

For internal use only.

Data Type: Integer

MovSzDBChkflag

For internal use only.

Data Type: Integer

Mousewait

Integer value indicating the amount of delay time for the mouse action to wait before processing additional clicks. The default value is 8.

Minimum = 0
Maximum = 16

```
bwEV(0).Mousewait = 8
```

This property may be set in the EGUI.INI file with **Mousewait=8.**

Data Type: Integer

Orange

Attribute place holder for the color orange. See Color System for more information.

Data Type: Long

Outlinecolr

The color to draw the outlines for all system objects. The default for this is black.

```
bwEV(0).Outlinecolr = bwEV(0).Black
```

This property may be set in the EGUI.INI file with **Outlinecolr=0** *(where 0 is the a color number 0-15).*

Data Type: Long

Red

Attribute place holder for the color red. See Color System for more information.

Data Type: Long

Scrollbarcolr

The color to paint scroll bars. The default for this is dark gray.

```
bwEV(0).Scrollbarcolr = bwEV(0).DarkGray
```

This property may be set in the **EGUI.INI** file with **Scrollbarcolr=8** *(where 8 is the a color number 0-15).*

Data Type: Long

Setpaletteflag

Flag which indicates if the EGUI internal palette configuration should be set on start up. By default this flag is set to True (-1).

```
bwEV(0).Setpaletteflag = -1
```

This property may be set in the **EGUI.INI** file with **Setpaletteflag=-1.**

Data Type: Integer

Statuscolr

The color to display pull down menu list items which are disabled. The default for this is gray.

```
bwEV(0).Statuscolr = bwEV(0).Gray
```

This property may be set in the **EGUI.INI** file with **Statuscolr=7** *(where 7 is the a color number 0-15).*

Data Type: Long

Titlebarcolr

The background color of a dialog box title bar. The default for this is blue.

```
bwEV(0).Titlebarcolr = bwEV(0).Blue
```

This property may be set in the **EGUI.INI** file with **Titlebarcolr=1** *(where 12 is the a color number 0-15).*

Data Type: Long

Titlebartextcolr

The foreground color of a dialog box title bar text. The default for this is white.

```
bwEV(0).Titlebarcolr = bwEV(0).White
```

This property may be set in the **EGUI.INI** file with **Titlebartextcolr=15** *(where 15 is the a color number 0-15).*

Data Type: Long

ViewportHgt

The current screen mode height. This information is set by the system and is readable but should <u>NOT</u> be reset.

Data Type: Integer

ViewportWid

The current screen mode width. This information is set by the system and is readable but should **NOT** be reset.

Data Type: Integer

Vportx1offset

The current screen mode horizontal offset. A viewport offset is used when and application has been designed to operate in the standard VGA 640x480 mode and is being used in a SuperVGA (800x600, etc.) Mode. See Display Modes and Viewports for more information. **Note:** This property is NOT supported in the Standard Video Driver Library.

Data Type: Integer

Vporty1offset

The current screen mode vertical offset. A viewport offset is used when and application has been designed to operate in the standard VGA 640x480 mode and is being used in a SuperVGA (800x600, etc.) Mode. See Display Modes and Viewports for more information. **Note:** This property is NOT supported in the Standard Video Driver Library.

Data Type: Integer

White

Attribute place holder for the color white. See Color System for more information.

Data Type: Long

Workspacecolr

For future development currently not supported.

Data Type: Long

Yellow

Attribute place holder for the color yellow. See Color System for more information.

Data Type: Long

Control Properties

Chapter 11

accelkey *Property*

Applies To

gCommandButton% and gChkOptBox%.

Description

Sets the key to be used as an accelerator key and the position to underline the character. **Note:** See *Chapter 9 Attaching Code to Controls* for more information about accelerator keys.

Usage

bwBT(0).accelkey = ***charrnum***

Remarks

Use this property to select a character to be used as an accelerator key.

charrnum

Integer value indicating the character position to use as an accelerator key.

Example: The letter "x" will be underlined and used as an accelerator key. The property value should be set to 9, because "x" is the ninth character in the string.

```
bwBT(0).accelkey = 9
Text$ = "Check Box One"
```

Data Type

Integer

active *Property*

Applies To gChkOptBox%.

Description Sets or returns the current state of the control.

Usage **bwBT(0).active** = ***setting%***

Remarks Use this property to set or get the state of a Check Box or Option Button control.

Note: Because Controls share property memory, this property could change when the focus is moved to a different control. So it is recommended that a Permanent Storage Variable be created to hold this value after getting it.
See the example in **Chapter 12** under **gChkOptBox%** *Control Function* for more information.

setting% Integer value indicating the state of the control.

True (-1) = Active
False (0) = Inactive

Data Type Integer

actsusflag *Property*

Applies To gChkOptBox%.

Description Suspense control state toggling if active (True).

Usage **bwBT(0).actsusflag =** ***setting%***

Remarks This property will suspend a Check Box or Option Button control from toggling to an Inactive state once the control is set to an Active state. This is the default configuration for an Option Button when placed in a Group by the EGUI Generator, and is optional for a check box.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

addobj *Property*

Applies To	gChkOptBox%, gComboBox%, gCommandButton%, gEditBox%, gHorzScrollBar%, gVertScrollBar% and gPullDownMenu%.
Description	Adds a Control Object to the Object Manager List.
Usage	**bwBT(0).addobj =** ***setting%*** ***(shown for Command Button)***
Remarks	Control Objects must be registered with the Object Manager List prior to operation of the Control. This property is usually set to False in the Dialog Box Property Section of the **DBFormat** and to True in the Draw Dialog Box Section, this will cause the control to be registered when the Dialog Box is drawn. A Control may only be registered once per Dialog Box and subsequent calls to add the same Control will be ignored. **Note:** The Control will be removed from the Object Manager List when **gRemoveDialogBox%** is called.

setting%	Integer value indicating the state of the flag. True (-1) = Add Object False (0) = Do Not Add Object

Data Type	Integer

addobjflag *Property*

Applies To gObjManager%.

Description Passes action instructions to the Object Manager.

Usage **bwOL(0).addobjflag =** ***setting%***

Remarks The Object Manager controls all the object controls in the EGUI System. To manipulate an object on the Object List, you send the Object Manager action instructions which are passed by this property.

setting% Integer value indicating which action to perform on the Object List. See **gObjManager%** *Control Function* in **Chapter 12** for a complete list of the available actions.

Data Type Integer

aptr *Property*

Applies To gListBox%.

Description Sets and returns the index number of the currently selected item in a list.

Usage **bwLB(0).aptr** = ***setting%***

Remarks Use this property to point to an item in a List Box list. When the List box Control loses the focus it will return the index to the currently selected item in this property. You may also pole this information while the List Box has the focus by setting the **bwLB(0).status** property to True. See **Chapter 9** *Attaching Code to Controls* for more information on using the **status** property.

Note: Because Controls share property memory, this property could change when the focus is moved to a different control. So it is recommended that a Permanent Storage Variable be created to hold this value after getting it.
See the example in **Chapter 12** under **gListBox%** *Control Function* for more information.

setting%	Integer value indicating the index number of the currently selected item in a list.

Note: If the **bwLB(0).tagflag** property is set to True this property is invalid, and you must scan the list to get a list of selected items.

Data Type Integer

See Also tagflag, rptr and dptr

assignobj *Property*

Applies To gCommandButton%.

Description Sets the object number to move the focus to after a button click.

Usage **bwBT(0).assignobj =** ***setting%***

Remarks Use this property when you want the focus to move or return to a different control after a button click.

setting%	Integer value indicating the object number to move the focus to.

Data Type Integer

autotab *Property*

Applies To	gEditBox%.
Description	Moves the focus to the next object in the Tab Order after the last character is entered in an Edit Box Control.
Usage	**bwEC(0).autotab =** ***setting%***

Remarks

Set this property to True (-1) when you want to move the focus to the next object in the Tab Order. This is useful in a application where data is being entered into several fields and you want to let the application move the focus to the next field after the user has enter all the information in the previous field.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

border *Property*

Applies To gCommandButton%, gComboBox%, gEditBox%, gChkOptBox%.

Description Draws a border around all or a portion of a control.

Usage **bwEC(0).border =** ***setting%***
(shown as Edit Box)

Remarks Set this property to True (-1) when you want to draw a border around a control. The border color is set by the environment property **bwEV(0).Outlinecolr**.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

button *Property*

Applies To gChkOptBox%.

Description Configures the Check/Option Box Control to act like either a Check Box or an Option Button.

Usage **bwBT(0).button =** ***setting%***

Remarks Because the Check and Option Button processes are so similar the Check/Option Box Control performs a dual role as both a Check Box and an Option Button. This property will configure the control for the desired process.

Note: When using this control configured as an Option Button there should only be one active button at a time in a group. This process is not supported directly by the library however it can be created. In fact when building forms in the EGUI Generator the generator will produce the code needed to do the grouping process and no further coding will be needed. To understand this process, create different sets of Option Button Group with the Form Generator and study the code it produces.

setting% Integer value indicating the state of the flag.

0 = Check Box
1 = Option Button

Data Type Integer

buttontype *Property*

Applies To gCommandButton% and gChkOptBox%.

Description Sets the style of a Command Button or a Check Box.

Usage **bwBT(0).buttontype** = *setting%*

Remarks The style of a Command Button or Check Box is mostly a matter of personal choice. Both controls continue to function in the same manner with the only difference being the appearance of the control. This feature could be used to signify different selection methods or simply as a cosmetic function. **Note:** This property has no effect on an Option Button.

setting% Integer value indicating the style.

Command Button (gCommandButton%)

0 = Standard
1 = Sunken
2 = Raised
3 = MS Windows Type Button

Check Box (gChkOptButton%)

0 = Check Mark
1 = Sunken Box
2 = Raised Box
3 = Large **X**

Data Type Integer

charhgt *Property*

Applies To gListBox%.

Description Sets and returns the number of rows to display in a List Box Control.

Usage **bwLB(0).charhgt =** ***setting%***

Remarks Use this property to set the number of rows for the List Box Control to display. Also the character height in the EGUI System currently only supports 8, 14 and 16 pixel high fonts. When different fonts are selected, (with the **sysfontnum** property), the List Box Control is resized so that the current number of rows stays the same. You may wish to readjust the number of rows to keep the control size approximately the same. **Important:** All Controls Boundaries should stay inside of the Dialog Box Border. Be sure NOT to set the number of rows value to high, so that the List Box Control will be drawn outside of the Dialog Box Border.

setting% Integer value indicating the number of rows to display. Depending on the font selected, if a Vertical Scroll Bar is being used try not to set this value to low so that the scroll bar may function properly. Minimum setting for a system font number 0 is 3.

Data Type Integer

charwid *Property*

Applies To	gListBox%.
Description	Sets and returns the number of columns to display in a List Box Control.
Usage	**bwLB(0).charwid** = ***setting%***
Remarks	Use this property to set the number of columns for the List Box Control to display. Also the character width in the EGUI System currently only supports 8 pixel wide fonts. **Important:** All Controls Boundaries should stay inside of the Dialog Box Border. Be sure NOT to set the number of columns value to high, so that the List Box Control will be drawn outside of the Dialog Box Border.

setting%	Integer value indicating the number of columns to display. If a Horizontal Scroll Bar is being used try not to set this value to low so that the scroll bar may function properly.

Data Type Integer

clearflag *Property*

Applies To gListBox%.

Description Sets a flag indicating for the List Box Control NOT to clear the highlighted item when losing the focus.

Usage **bwLB(0).clearflag =** ***setting%***

Remarks Use this property when you wish to leave an item which has been selected in a list box, highlighted when moving the focus to another control.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

col *Property*

Applies To gComboBox% and gEditBox%.

Description Sets and returns the left most column position to place a Control Object at.

Usage **bwEC(0).col** = ***setting%***

Remarks The Edit Box and Combo Box Controls use a mixed coordinate system to place their location on the display. Column is any value between 1 and 79, however it is usually added to the Dialog Box Upper Left Corner Offset. Note that the Dialog Box setting is in pixel values so there is a miner conversion involved in setting the column value. Below is the standard formula for setting this value.

setting% Integer value indicating the column to place a control. Use the formula below to calculate this value.

bwEC(0).col = ((x1% \ 8) + 11)

x1% = the upper left x value of the Dialog
8 = the standard font width in pixels
11 = the number of columns to offset inside of the Dialog Box to the right. This number should be replaced with your column number, any number between (1-79).

Also note that **Integer division** (\) should be used not floating point division (/).

See Also **row**

Data Type Integer

combo *Property*

Applies To gComboBox%.

Description Sets a flag indicating the gComboBox% Control should function like a Combo Box.

Usage **bwEC(0).combo** = ***setting%***

Remarks There are two types of Combo Boxes available, a *Combo Box* and a *Drop Down Box*. A *Combo Box* allows the user to type an entry in the Combo Box or pick and item from the Drop Down List. A *Drop Down Box* only allows the user to select and item from the Drop Down List. One exception to this is if the **bwEC(0).noeditflag** property is set to True no editing is allowed in a *Combo Box* or a *Drop Down Box.*

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

See Also **dropdown, noeditflag**

Data Type Integer

dblclkflag *Property*

Applies To gListBox%.

Description Sets a flag indicating that the gListBox% Control should process a mouse double click when selecting an item from the list.

Usage **bwLB(0).dblclkflag =** ***setting%***

Remarks If this property is set to True (-1) then the List Box Control will process a mouse double click when selecting an item from the list. This is the same as selecting the item with the keyboard and pressing the ENTER key.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

See Also **mousewait** (Environment Property)

Data Type Integer

dbx1, dby1, dbx2, dby2 *Property*

Applies To gBuildDialogBox%.

Description Sets the Upper Left and Lower Right corner coordinates in pixels of a Dialog Box Control.

Usage

bwDB(0).dbx1 = *setting%*
bwDB(0).dby1 = *setting%*
bwDB(0).dbx2 = *setting%*
bwDB(0).dby2 = *setting%*

Remarks A Dialog Box Control is drawn to the display with the values set in these properties. **Important:** The **x** values must be set to an 8 pixel increment (i.e. 0, 8, 16, 32, etc.). The **y** values may be set to any row number within the legal limits of the selected Display Mode (i.e. **Mode 12-640x480** allowable **y** values are 0-479).

setting%	Integer value indicating the pixel location to draw a Dialog Box Control.

See Also **displaymode** (Environment Property)

Data Type Integer

dcolr *Property*

Applies To

BuildDialogBox%,gComboBox%,gCommandButton%,gEditBox%, gListBox%, gChkOptBox%.

Description Sets the drawing color of a Control's background.

Usage **bwDB(0).dcolr =** ***setting&***
(shown for Dialog Box)

Remarks If this property is set to zero (0) then the default control color is used to draw the background of that control. This property is used to override the default value with a custom value for that specific control. **Note:** This property is set using a *Multi Color Value* for a **gChkOptBox%** Control where the foreground value is used to modify the color of the Option Button Label and the background is used to modify the background color of the Option Button. **Important:** All other controls only use a *Single Color Value* for setting the background of that control. See **Color System** in **Chapter 3** for more information.

setting&	Long Integer value indicating a Control's background color.

See Also **dialogboxcolr** (Environment Property)
controlcolr (Environment Property)

Data Type Long

defbutbox *Property*

Applies To gCommandButton%.

Description Sets a flag which indicates that the Command Button Control should draw a *Default Button Box* around the button.

Usage **bwBT(0).defbutbox** = ***setting%***

Remarks A *Default Button Box* indicates to the user which command button is active if the ENTER key is pressed. If this flag is True (-1), when a command button gets the focus it will draw a box around the edge of the button. When the button loses the focus it will remove the box.

Important: If you are using a Default Button Box for one button you should use it for all buttons on that dialog box. Remember when a command button has the focus the ENTER key will activate what ever process is attached to that button. So the default box should move to the command button which has the focus. When a control, other than a command button, has the focus you should turn on the Default Button Box around the button which is the default. To do this set up to local processes. One to turn the default on and another to turn it off. See the sample code EGUIINST.BAS in the sample directories for and example on how to use this property.

setting% Integer value indicating the state of the property.

True (-1) = Active
False (0) = Inactive

Data Type Integer

depress *Property*

Applies To gCommandButton%.

Description Sets and returns the mode of a Command Button.

Usage **bwBT(0).depress =** ***setting%***

Remarks Use this property to select the mode of a Command Button. The two modes available for a Command Button are **Up** and **Down**. This property should be used when you wish to use Command Buttons in a PushOn-PushOff type configuration., much like is used on a Tool Bar. This action is not directly supported in the library and requires some additional coding. See **Building a Tool Bar** in **Chapter 9** for more information.

setting%	Integer value indicating the mode of the control. -1 = Control Up -2 = Control Down

Data Type Integer

dlen *Property*

Applies To gComboBox% and gEditBox%.

Description Sets and returns the display length of a Control.

Usage **bwEC(0).dlen** = ***setting%***

Remarks Use this property to set the number of characters to display in a control. For a Combo Box Control this only affects the Edit Box portion of the control. Note that the Edit Box Control may be scrolled left and right so it is not absolutely necessary to set the display length to the actual string length.

setting%	Integer value indicating the number of characters to display.

See Also **slen**

Data Type Integer

dptr *Property*

Applies To gComboBox%, gEditBox% and gListBox%.

Description Sets and returns an index to the left most character being viewed in the control.

Usage **bwEC(0).dptr =** ***setting%***
(shown as Edit Box)

Remarks While scrolling left and right in a control, an index pointer is maintained of the left most character being displayed. This property may be used to manage this index.

Note: Because Controls share property memory, this property could change when the focus is moved to a different control. So it is recommended that a Permanent Storage Variable be created to hold this value after getting it.

See the example in **Chapter 12** under **gEditBox%** *Control Function* for more information.

setting%	Integer value indicating the index number.

See Also **eptr, aptr, rptr**

Data Type Integer

dropdown *Property*

Applies To gComboBox%.

Description Sets a flag indicating the gComboBox% Control should function like a Drop Down Box.

Usage **bwEC(0).dropdown =** ***setting%***

Remarks There are to different types of Combo Boxes available, a Combo Box and a Drop Down Box. A Combo Box allows the user to type an entry in the Combo Box or pick and item from the Drop Down List. A Drop Down Box only allows the user to select and item from the Drop Down List. One exception is if the **bwEC(0).noeditflag** property is set to True, then no editing is allowed in a Combo Box or a Drop Down Box.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

See Also **combo, noeditflag**

Data Type Integer

editflag *Property*

Applies To gListBox%.

Description Sets a flag indicating the gListBox% Control should function like a Multi Line Edit Box Control.

Usage **bwLB(0).editflag =** ***setting%***

Remarks The Edit Box Control may only be used to edit a single line of text. When set to True (-1), this property will combine the Edit Box Control and List Box Control to make a Multi Line Edit Box Control. This control will function much like a normal text line editor and it has two editing features **Insert Line (Ctrl-L)** and **Delete Line (Ctrl-D)** .

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

elmwid *Property*

Applies To gListBox%.

Description Sets and returns the element width in characters for an item in the List Box Control.

Usage **bwLB(0).elmwid** = ***setting%***

Remarks The List Box Control may be scrolled left and right. By setting the **elmwid** property to a value higher then the **charwid** property the list box will automatically allow left and right scrolling. You may use the left and right arrow keys or optionally add a Horizontal Scroll Bar for controlling the scroll process.

setting% Integer value indicating the width in characters of a List Box item.

Max = 255
Min = 1

See Also **hsbflag, charwid**

Data Type Integer

enable *Property*

Applies To	gPullDownMenu%, gCommandButton%, gEditBox%, gListBox%, gComboBox%,gChkOptBox%, gHorzScrollBar%,gVertScrollBar% and ObjManager%.
Description	Sets a flag which indicates if a Control Object is enabled or disabled.
Usage	**bwLB(0).enable** = ***setting%*** ***(shown as List Box)***
Remarks	A Control Object may be Enabled or Disabled after it has been registered with the Object Manager. When a Control is Enabled it will function normally. If a Control is Disabled it will ignore any messages or events that are sent to it. An exception to this rule is that a Dialog Box and Pull Down Menu Control are always enabled, so this property setting is ignored. Also the Object Manager uses this property to indicate when first adding an Object to the Object List if it should be enable or disabled.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

See Also	**paintobj, addobj, objidnum**
Data Type	Integer

eptr *Property*

Applies To gComboBox% and gEditBox%.

Description Sets and returns an index to the current cursor position in a control.

Usage **bwEC(0).eptr** = ***setting%***

Remarks While editing text in and Edit Box an index pointer to the current cursor position is maintained. This index will be return in this property.

Note: Because Controls share property memory, this property could change when the focus is moved to a different control. So it is recommended that a Permanent Storage Variable be created to hold this value after getting it.
See the example in **Chapter 12** under **gEditBox%** *Control Function* for more information.

setting%	Integer value indicating the index number of current cursor position.

See Also **dptr**

Data Type Integer

format *Property*

Applies To gComboBox% and gEditBox%.

Description Sets a value indicating that a predefined format should be used with this control.

Usage **bwEC(0).format** = ***setting%***

Remarks There are 5 predefined entry formats that may be selected to be used with the Edit Box Control. These formats will mask the entry input in the control and only allow specific characters to be enter.

setting% Integer value indicating which format to use.

1 = Social Security Number
2 = Extended Zip Code
3 = Phone Number
4 = Area Code & Phone Number
5 = Date

Note: There is currently no support for a User Defined Format.

Data Type Integer

frame *Property*

Applies To gHorzScrollBar% and gVertScrollBar%.

Description Sets and returns the value of a scroll bar frame.

Usage **bwSB(0).frame =** ***setting!***

Remarks A Scroll Bar frame property is used to set the aspect ratio of a Scroll Bar. This property should usually be set to the same value as the **largechange** property.

setting!	Single Integer value indicating the aspect ratio of a scroll bar.

See Also **largechange, smallchange, min, max**

Data Type Single

headercolr *Property*

Applies To gListBox%.

Description Sets the color of the Header Text in a List Box Control.

Usage **bwLB(0).headercolr =** ***setting&***

Remarks This is a *Multi Color Value* used to set the color of the List Box Header Text. See **Color System** in **Chapter 3** for more information.

setting& Long Integer value indicating the color of the header text.

See Also **headerflag, headerfont**

Data Type Long

headerflag *Property*

Applies To gListBox%.

Description Sets a flag indicating if a List Box Header is to be drawn.

Usage **bwLB(0).headerflag =** ***setting%***

Remarks If you wish to display a Header Row on top of a List Box set this property to True (-1). See **gListBox%** *Control Function* in Chapter 12 for information on the Header Text.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

See Also **headercolr, headerfont**

Data Type Integer

headerfont *Property*

Applies To

gListBox%.

Description

Sets a value indicating which font number to use when displaying a List Box Header.

Usage

bwLB(0).headerfont = ***setting%***

Remarks

If you are displaying a List Box Header you may select a different font than the one used to display List Box Items. Use this property to select one of the following fonts.

setting% Integer value indicating which font number to use.

0 = 8x16	'**Bold**
1 = 8x14	'**Bold**
2 = 8x14	'Normal
3 = 8x14	'*Italic*
4 = 8x8	'**Bold**
5 = 8x8	'Normal
6 = 8x8	'*Italic*

See Also

headercolr, headerflag

Data Type

Integer

hlinecolr *Property*

Applies To	gComboBox% and gListBox%.
Description	Sets the color of the List Box Control Divider Lines.
Usage	**bwLB(0).hlinecolr =** ***setting&***
Remarks	This is a *Single Color Value* used to set the color of the List Box Divider Lines. See **Color System in Chapter 3** for more information. **Note:** The lines first must be turned on using **hlineflag**.

setting&	Long Integer value indicating the color of the divider lines.

See Also	**hlineflag**
Data Type	Long

hlineflag *Property*

Applies To gComboBox% and gListBox%.

Description Sets a flag indicating that the Divider Lines should be turned on in the List Box Control.

Usage **bwLB(0).hlineflag** = *setting%*

Remarks When this property is set to True (-1) the List Box Divider Lines will be drawn below each item row in the list and a shadowed box will be drawn around this information.

Note: If the **bwLB(0).editflag** property is set to True this property is ignored.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

See Also **hlinecolr**

Data Type Integer

hpagesz *Property*

Applies To gComboBox% and gListBox%.

Description Sets the horizontal page movement size of the List Box Control.

Usage **bwLB(0).hpagesz** = ***setting%***

Remarks This feature is similar to the **largechange** property with a Horizontal Scroll Bar. It sets the aspect distance the page should scroll left and right when the Scroll Bar is clicked. This property should normally be set to the value 1.

Important: Make sure that a Permanent Variable is setup for saving and restoring the **bwLB(0).dptr** property before doing horizontal scrolling.

setting%	Integer value indicating the horizontal move distance.

See Also **dptr**

Data Type Integer

hsbflag *Property*

Applies To gComboBox% and gListBox%.

Description Sets a flag indicating that a Horizontal Scroll Bar should be displayed in a List Box Control.

Usage **bwLB(0).hsbflag =** ***setting%***

Remarks This property will turn a Horizontal Scroll Bar on for a List Box Control. Note that the items in the list must extend past the left edge of the list box to produce horizontal scrolling.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

icon *Property*

Applies To gCommandButton%.

Description A flag indicating that an EGUI Icon File should be loaded and displayed on a Command Button.

Usage **bwBT(0).icon =** ***setting%***

Remarks This property when set to True (-1) tells the Command Button Control to load and display the EGUI Icon which is listed in the **bwBT(0).iconfile** property. If the Icon File cannot be located or it is and invalid file format, this property will be ignored.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

See Also **iconfile, iconx1off, icony1off**

Data Type Integer

iconfile *Property*

Applies To gCommandButton%.

Description Sets an EGUI Icon File Name to be loaded and displayed on a Command Button.

Usage **bwBT(0).iconfile** = ***setting$***

Remarks When the **bwBT(0).icon** property is set to True (-1) , the Icon File Name set in this property will be loaded. You may optional set a drive and directory. The EGUI System Variable **IconPath$** is set at application startup time to the location of the system Icon Files. It is recommended that all Icon Files be stored in this location for better management of Icons. Below is and example on how to use the system path information.

setting$	String value indicating the Icon File to load. Optional drive and directory is permitted.

System Path Example:

```
bwBT(0).iconfile = IconPath$+'\MYICON.ICN'
```

See Also **icon, iconxloff, iconyloff**

Data Type String

iconxloff & iconyloff *Property*

Applies To gCommandButton%.

Description Sets a Command Button Icon x,y offset.

Usage
bwBT(0).iconxloff = ***setting%***
bwBT(0).iconyloff = ***setting%***

Remarks When the **bwBT(0).icon** property is set to True (-1) , an Icon File is loaded and displayed on the Command Button. The Icon x,y offset is the offset distance in pixels from the upper left corner of the Command Button. The default offset for both x & y is 5, however you may adjust this offset so that an icon may be placed any where on the button. **Note:** the Load routine does not clip the icon, so be careful not to make the offset so high that the icon is displayed outside of the Command Button Border.

setting% Integer value indicating the offset distance in pixels to place the icon at.

See Also **icon, iconfile**

Data Type Integer

intflag *Property*

Applies To gListBox%.

Description Sets and returns the List Box Control's Initialization Mode values.

Usage **bwBT(0).intflag** = *setting%*

Remarks The List Box Mode Initialization flag is used to help control different configurations of the List Box Control, such as a Paint or Update Event and when it Gets and Loses the Focus. Note that this property changes return codes for different configurations.

setting% Integer value indicating the which mode to select.

Passed To:

-256 = Standard Initialization with List Box Prompt. The **List Box Prompt** is a dotted line around the inside of the List Box Control Area. When a prompt is displayed no item has been selected yet. Press the DOWN ARROW key to move past the prompt.

-255 = Bypass the List Box Prompt and select the item that is indexed in the **bwLB(0).aptr** property.

-254 = Highlight the list Item that is indexed in the **bwLB(0).aptr** property on an update process. This mode is ONLY available on a update process.

Returned From:

-2 = On a Mouse Click outside of the List Box Control Area.

-3	=	On a Paint, Update or Add Object Event.
-4	=	On a Status Event.

Data Type Integer

largechange *Property*

Applies To gHorzScrollBar% and gVertScrollBar%.

Description Determines the amount of change to report in a scroll bar control when the user clicks on the scroll bar. The **value** property increases or decreases by this amount.

Usage **bwSB(0).largechange** = ***setting!***

Remarks The amount may be any value between 1 and 32,767 but should be no larger than the difference between **min** and **max** properties. See **gHorzScrollBar%** in **Chapter 12** for more information.

setting!	Single Integer value indicating the change amount.

Data Type Single

logicflag *Property*

Applies To gCommandButton%.

Description Determines if a Command Button has a Logical Operator applied during the Down Event.

Usage **bwBT(0).logicflag =** ***setting%***

Remarks This value may be between 0 and 4 and selects a Logical Operation to be applied to the Command Button Control when it is being depressed.

setting% Integer value indicating the Logical Operation.

0 = PSET
1 = PRESET
2 = AND
3 = OR
4 = XOR

Note: The **XOR** operation is usually used on a Tool Bar to enhance the depressed appearance.

Data Type Integer

max *Property*

Applies To gHorzScrollBar% and gVertScrollBar%.

Description Determines the scroll bar's maximum position value.

Usage **bwSB(0).max =** ***setting%***

Remarks This value may be between -32,768 and 32,767. The default value is 100. The scroll bar button is at the maximum value when it is at the bottom of a vertical scroll bar and the right most side of the horizontal scroll bar.

setting%	Integer value indicating the maximum position of a Scroll Bar Control.

Data Type Integer

menucolr *Property*

Applies To gPullDownMenu%.

Description Determines the background color of a Pull Down Menu.

Usage **bwPD(0).menucolr =** ***setting&***

Remarks This is a *Single Color Value* which is used to paint the menu bar and pull down list background. See **Color System** in **Chapter 3** for more information.

setting&	Long Integer value indicating the control background color.

Data Type Long

menutextcolr *Property*

Applies To gPullDownMenu%.

Description Determines the foreground color of a Pull Down Menu.

Usage **bwPD(0).menutextcolr =** ***setting&***

Remarks This is a *Single Color Value* which is used to paint the menu bar and pull down list foreground (Characters). See **Color System** in **Chapter 3** for more information.

setting&	Long Integer value indicating the control foreground color.

Data Type Long

menxl, menyl, menx2, meny2 *Property*

Applies To gPullDownMenu%.

Description Determines the upper left and lower right corners of the Pull Down Menu Bar.

Usage

bwPD(0).menxl = ***setting%***
bwPD(0).menyl = ***setting%***
bwPD(0).menx2 = ***setting%***
bwPD(0).meny2 = ***setting%***

Remarks These coordinates set the location of a menu bar on a dialog box. To determine the correct values for these properties use the information in the example code. **Note:** These properties are not available in the EGUI Generator only the Library.

setting%	Integer value indicating the location of the pull down menu bar in pixels.

Data Type Integer

(see next page for example)

The following example shows how to determine the correct pull down menu coordinates. Note this code is built automatically in the EGUI Generator at Function Build Time.

Example

```
bwPD(0).menx1  =  (bwDB(0).dbx1+bwEV(0).Borderwid)
bwPD(0).meny1  =  (bwDB(0).dby1+bwTitleBarHgt%+bwEV(0).Borderwid)
bwPD(0).menx2  =  (bwDB(0).dbx2 - bwEV(0).Borderwid)
bwPD(0).meny2  =  -1
```

min *Property*

Applies To gHorzScrollBar% and gVertScrollBar%.

Description Determines the scroll bar's minimum position value.

Usage **bwSB(0).max** = ***setting%***

Remarks This value may be between -32,768 and 32,767. The default value is 0. The scroll bar button is at the minimum value when it is at the top of a vertical scroll bar and the left most side of the horizontal scroll bar.

setting%	Integer value indicating the minimum position of a Scroll Bar Control.

Data Type Integer

movetoflag *Property*

Applies To gComboBox% and gListBox%.

Description Sets a flag to indicate that a List Box Control will move to the next item in the list which starts with the character pressed.

Usage **bwLB(0).movetoflag** = ***setting%***

Remarks This property when set to True (-1), will cause the List Box Control to move to the next item in the list that starts with the character that was pressed by the user. If no item on the list starts with that character the request is ignored. After the last item in the list is reached the process loops back to the top.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

noborderflag *Property*

Applies To gListBox%.

Description Sets a flag to indicate that a List Box Control will be displayed with no border.

Usage **bwLB(0).noborderflag =** ***setting%***

Remarks This property when set to True (-1), will cause the List Box Control not to display a border. By default a List Box Control always has a border.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

noeditflag *Property*

Applies To gComboBox% and gEditBox%.

Description Sets a flag to indicate that a Edit Box Control will NOT allow editing, display only.

Usage **bwEC(0).noeditflag =** ***setting%***

Remarks This property when set to True (-1), will cause the Edit Box Control not to allow editing of it's text. This is a display only mode for the edit box. Note that scrolling events are still enabled even though you may not edit the text.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

nonborderflag *Property*

Applies To gBuildDialogBox%.

Description Sets a flag to indicate that a Dialog Box Control will be displayed with a thin border.

Usage **bwDB(0).nonborderflag =** ***setting%***

Remarks This property when set to True (-1), will cause the Dialog Box Control to be displayed with a thin border. By default a Dialog Box Control always has a wide border. The border around a dialog box is used to indicate types of dialog boxes. In future development the wide border around a dialog box will be used to locate size handles. A thin border will be a non-sizeable box. See **Window Types** under **Chapter 8** for more information.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

noncloseflag *Property*

Applies To gBuildDialogBox%.

Description Sets a flag to indicate that a Dialog Box Control will be displayed without a Close Button.

Usage **bwDB(0).noncloseflag =** ***setting%***

Remarks This property when set to True (-1), will cause the Dialog Box Control to be displayed without a Close Button. The Close Button is used to remove the dialog box from the screen when the user is finished using it. During development time this is the only method for removing a dialog box until code is attached to some other event which allows you to close the box. Thus it is recommended that a Close Button always exist unless you are sure you will not need it.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

nonmoveflag *Property*

Applies To gBuildDialogBox%.

Description Sets a flag to indicate that a Dialog Box Control will not be movable.

Usage **bwDB(0).nonmoveflag =** ***setting%***

Remarks This property when set to True (-1), will cause the Dialog Box Control to be a non-movable window. By default a Dialog Box Control is a movable window. When a Dialog Box is a movable window it will have a border around the Title Bar. A non-movable window does not have a border. To move a Movable Window place the mouse cursor inside the Title Bar Box and press the left mouse button. While holding the left mouse button down reposition the box where you wish and release the button.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

novsbflag *Property*

Applies To	gComboBox% and gListBox%.
Description	Sets a flag to indicate that a List Box Control will not display a vertical scroll bar.
Usage	**bwLB(0).novsbflag =** ***setting%***
Remarks	This property when set to True (-1), will cause the List Box Control to be displayed without a vertical scroll bar. By default a List Box has a vertical scroll bar.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

num2disp *Property*

Applies To gComboBox%.

Description Sets and returns the number of rows to display in a Combo Box Control.

Usage **bwEC(0).num2disp** = ***setting%***

Remarks Use this property to set the number of rows for the Combo Box Control to display. Also the character height in the EGUI System currently only supports 8, 14 and 16 pixel high fonts. When different fonts are selected, (with the **sysfontnum** property), the Combo Box Control is resized so that the current number of rows stays the same. You may wish to readjust the number of rows to keep the control size approximately the same.

setting%	Integer value indicating the number of rows to display. Depending on the font selected, if a Vertical Scroll Bar is being used, do not set this value to low as the scroll bar may not function properly. Minimum setting for a system font number 0 is 3.

Data Type Integer

numflag *Property*

Applies To gComboBox% and gEditBox%.

Description Sets a flag which indicates that an Edit Box Control will only accept numeric values.

Usage **bwEC(0).numflag** = ***setting%***

Remarks Set this property to True (-1) to make the Edit Box Control only allow the entry of numeric characters. When this property is False alphanumeric characters are allowed.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

objheight *Property*

Applies To gCommandButton%, gHorzScrollBar% and gVertScrollBar%.

Description Sets and returns the height of a Control Object.

Usage **bwBT(0).objheight =** ***setting%***
(shown as Command Button)

Remarks This value is set in the EGUI Generator at design time, however it may be readjusted in code as needed. This is a pixel value which determines the physical height of the Control. **Note**: The height of a Horizontal Scroll Bar is fixed by the system and is not adjustable.

setting%	Integer value indicating the height of the Control Object in pixels.

Data Type Integer

objid *Property*

Applies To

gBuildDialogBox%, gPullDownMenu%, gCommandButton%, gEditBox%, gListBox%, gChkOptBox%, gHorzScrollBar% and gVertScrollBar%,

Description

Sets a flag which indicates that a Control Object should show its active control object indicator when it gets the focus.

Usage

bwLB(0).objid = ***setting%***
(shown as List Box)

Remarks

If this flag is set to True (-1) then the Control Object will display it's active object indicator when it gets the focus. If the flag is False (0) then no indicator will be shown. An active control object indicator can be an insertion point, a highlight or a dotted line.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type

Integer

objidnum *Property*

Applies To gBuildDialogBox%, gPullDownMenu%, gCommandButton%, gEditBox%, gListBox%, gChkOptBox%, gHorzScrollBar%, gVertScrollBar% and gObjManager%.

Description Returns and sets the Active Control Object ID Number. (The Object Handle).

Usage **bwLB(0).objidnum** = ***setting%***
(shown as List Box)

Remarks The Object Manager assigns each Control Object a unique object handle at the time it is registered. This handle, also called the Control Object ID Number, is used to perform different tasks on specific Controls. Note that most procedures in the EGUI system do not require this information.

Note: Because Controls share property memory, this property could change when the focus is moved to a different control. So it is recommended that a Permanent Storage Variable be created to hold this value after getting it.
The best time to get this information is immediately after a Control has been registered (added to the object list).

setting%	Integer value indicating the handle or Control Object ID Number of the current object.

Data Type Integer

objwidth *Property*

Applies To gCommandButton%,gHorzScrollBar% and gVertScrollBar%.

Description Sets and returns the width of a Control Object.

Usage **bwBT(0).objwidth =** ***setting%***
(shown as Command Button)

Remarks This value is set in the EGUI Generator at design time, however it may be readjusted in code as needed. This is a pixel value which determines the physical width of the Control. **Note**: The width of a Vertical Scroll Bar is fixed by the system and is not adjustable.

setting%	Integer value indicating the width of the Control Object in pixels.

Data Type Integer

objxl, objyl, objx2, objy2 *Property*

Applies To gObjManager%.

Description Determines the upper left and lower right corners of a Control Object.

Usage

bwOL(0).objxl = ***setting%***
bwOL(0).objyl = ***setting%***
bwOL(0).objx2 = ***setting%***
bwOL(0).objy2 = ***setting%***

Remarks These coordinates set the location of a Control Object's Boundary at the time it is registered with the Object Manager. The upper left corner is represented by **objxl,objyl** and the lower right corner is represented by **objx2,objy2**. The EGUI System Control Objects take care of this process. The only time you will need to perform this process is when you are creating a Custom Control. See **Custom Controls** in **Appendix B** for more information.

setting%	Integer value indicating the location of the Control Object's Boundary in pixels.

Data Type Integer

oiflag *Property*

Applies To gComboBox% and gEditBox%.

Description Sets a flag to indicate that a Edit Box Control should be in Overstrike or Insert Mode.

Usage **bwEC(0).oiflag =** ***setting%***

Remarks This property when set to True (-1), will cause the Edit Box Control to be in Insert Mode. If set to False (0) it will be in Overstrike Mode.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

paintobj *Property*

Applies To	gBuildDialogBox%, gPullDownMenu%, gCommandButton%, gEditBox%, gListBox%, gChkOptBox%, gHorzScrollBar%, gVertScrollBar%.
Description	Sets a flag which indicates that a Control Object should perform a paint event.
Usage	**bwLB(0).paintobj =** ***setting%*** ***(shown as List Box)***
Remarks	When this property is set to True (-1) a Control Object performs a paint event, which draws the control to the display. This event usually happens when a Control Object is registered. Note that it is usually not necessary to have a paint event when an update event is requested. An **update** event differs from a **paint** event in that the paint event will paint the entire control (i.e. its border, scroll bars, etc.) and an update event will only paint the action of the control. For example an update of a List Box will only paint the items in the list and not the border or scroll bars.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

See Also	**addobj, objidnum, update**
Data Type	Integer

rate *Property*

Applies To gComboBox% and gEditBox%.

Description Sets and returns the Edit Box cursor blink rate value.

Usage **bwEC(0).rate =** ***setting%***

Remarks Use this property to set the rate of speed you wish the Edit Box cursor to blink at. This may be any value between 1 and 15, the default value is 4. Note if rate is set to 0 the cursor will stop blinking.

setting%	Integer value indicating the cursor blink rate.

Data Type Integer

reobjx1, reobjy1, reobjx2, reobjy2 *Property*

Applies To gObjManager%.

Description Determines new upper left and lower right corner coordinates for an existing Control Object's Boundary.

Usage

bwOL(0).reobjx1 = ***setting%***
bwOL(0).reobjy1 = ***setting%***
bwOL(0).reobjx2 = ***setting%***
bwOL(0).reobjy2 = ***setting%***

Remarks These properties are used to reposition a Control Object original boundary coordinates when repositioning the Control. See **Custom Controls** in **Appendix B** for more information.

setting%	Integer value indicating the new location of the Control Object's Boundary in pixels.

Data Type Integer

row *Property*

Applies To gComboBox% and gEditBox%.

Description Sets and returns the row position to place a Control Object.

Usage **bwEC(0).row =** ***setting%***

Remarks The Edit Box and Combo Box Controls use a mixed coordinate system to place their location on the display. Row is any value between (0-479) for Display Mode 12 and (0-349) for Display Mode 9, however it is usually added to the Dialog Box Upper Left Corner **Y** Offset. Below is the standard formula for setting this value.

setting% Integer value indicating the row to place a control. Use the formula below to calculate this value.

bwEC(0).row = (y1% + 34)

y1% = the upper left y value of the Dialog
34 = the number of rows to offset inside of the Dialog Box down

See Also **col**

Data Type Integer

rptr *Property*

Applies To gListBox%.

Description Sets and returns the index number of the currently selected item's row in a list.

Usage **bwLB(0).rptr** = ***setting%***

Remarks Use this property in combination with the **bwLB(0).aptr** property to point to an item in a List Box list. When the List box Control loses the focus it will return the index to the currently selected item's row in this property. You may also pole this information while the List Box has the focus by setting the **bwLB(0).status** property to True. See **Chapter 9** *Attaching Code to Controls* for more information on using the **status** property.

Note: Because Controls share property memory, this property could change when the focus is moved to a different control. So it is recommended that a Permanent Storage Variable be created to hold this value after getting it.
See the example in **Chapter 12** under **gListBox%** *Control Function* for more information.

setting% Integer value indicating the index number of the currently selected item's row in a list.

See Also **aptr, dptr**

Data Type Integer

shadowflag *Property*

Applies To gPullDownMenu%,gComboBox%,gEditBox%, gListBox%, gChkOptBox%.

Description Sets a flag which indicates that a shadow will be drawn around the Control.

Usage **bwEC(0).shadowflag =** ***setting%*** ***(shown for Edit Box)***

Remarks If this property is set to True (-1), when the Control gets a paint event it will draw a shadow around the edge of the control. The actual shadow appearance may vary between controls.

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

slen *Property*

Applies To gComboBox% and gEditBox%.

Description Sets and returns the actual string length of the text in an Edit Box Control. This length may be longer that the display length (**dlen**).

Usage **bwEC(0).slen =** ***setting%***

Remarks Use this property to set the actual number of characters that the Edit Box Control will allow to be edited. For a Combo Box Control this only effects the Edit Box portion of the control. Note that the Edit Box Control may be scrolled left and right so it is not absolutely necessary to set the display length to the actual string length, but the string length should always be as long or longer than the display length.

setting% Integer value indicating the actual text string length being edited. Maximum of 255 characters.

See Also **dlen**

Data Type Integer

smallchange *Property*

Applies To gHorzScrollBar% and gVertScrollBar%.

Description Determines the amount of change to report in a scroll bar control when the user clicks on a scroll button. The **value** property increases or decreases by this amount.

Usage **bwSB(0).smallchange =** ***setting!***

Remarks The amount may be any value between 1 and 32,767 but should be no larger than the difference between **min** and **max** properties and should be smaller than the **largechange** property. See **gHorzScrollBar%** in **Chapter 12** for more information.

setting!	Single Integer value indicating the change amount.

Data Type Single

sortflag *Property*

Applies To gComboBox% and gListBox%.

Description Sets a flag indicating the gListBox% Control should sort the list prior to a paint or update event.

Usage **bwLB(0).sortflag =** ***setting%***

Remarks This property when set to True (-1) will cause the List Box Control to sort it's list in ascending order prior to a paint or update event.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

status *Property*

Applies To gBuildDialogBox%, gPullDownMenu%, gCommandButton%, gEditBox%, gListBox%, gChkOptBox%.

Description Sets a flag which indicates that a Control Object should report it's current status while it has the focus.

Usage **bwLB(0).status =** ***setting%***
(shown as List Box)

Remarks A Control Object usually reports its status when it loses focus. When this flag is set to True you may trap the status of the Control as it is processing. For more information on using the status property see **Attaching Code to Control Chapter 9**.

setting%	Integer value indicating the state of the flag.
	True (-1) = Active False (0) = Inactive

Data Type Integer

sysfontnum *Property*

Applies To gBuildDialogBox%, gPullDownMenu%,gComboBox%,gEditBox%, gCommandButton%, gListBox%, gChkOptBox%.

Description Sets a flag which indicates which System Font Number should be used with the Control.

Usage **bwLB(0).sysfontnum =** ***setting%***
(shown as List Box)

Remarks A Control Object which displays fonts may use any of the available system fonts. This property will set the value of the font number to be used.

setting% Integer value indicating the font number to use. Valid values are 0 through 6.

0 = 8x16	**'Bold**
1 = 8x14	**'Bold**
2 = 8x14	'Normal
3 = 8x14	*'Italic*
4 = 8x8	**'Bold**
5 = 8x8	'Normal
6 = 8x8	*'Italic*

Data Type Integer

tabflag *Property*

Applies To gPullDownMenu%, gCommandButton%, gEditBox%, gListBox%, gComboBox%,gChkOptBox%, gHorzScrollBar%,gVertScrollBar% and ObjManager%.

Description Sets a flag which indicates if a Control Object's tab event is enabled or disabled.

Usage **bwLB(0).tabflag** = ***setting%***
(shown as List Box)

Remarks To move focus between Control Object with the keyboard you press the TAB key or SHIFT-TAB key. This event can be enabled and disabled by this property flag. Set the property to True (-1) to enable tabbing and false to disable tabbing.

setting%	Integer value indicating the state of the flag. True (-1) = Enabled False (0) = Disabled

See Also **paintobj, addobj, objidnum**, enable

Data Type Integer

tagflag *Property*

Applies To gListBox%.

Description Sets a flag indicating the List Box Control should allow the tagging of multiple items.

Usage **bwLB(0).tagflag =** ***setting%***

Remarks This property when set to True (-1) will cause the List Box Control to allow the tagging of multiple items. To tag and items either click on it with the left mouse button or use the select key (SPACE BAR). When an item is tagged a check mark will be placed in the item element at the last character position. To get the tagged items simply scan the element array for this character. The selection process functions like a toggle, so to unselect an item either click the mouse button again or press the select key again.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type Integer

title *Property*

Applies To gBuildDialogBox%.

Description Sets the Title of a Dialog Box.

Usage **bwDB(0).title =** ***setting$***

Remarks This property sets the Title of a Dialog Box. It may be a maximum of 60 characters, however make sure that the title will fit in the Dialog Box Title Bar or it may be displayed outside of the box. The Title is centered between the x1 and x2 dialog box borders by default.

setting$	String value indicating the dialog box title. Maximum of 60 characters.

Data Type String

titlebaroffset *Property*

Applies To	gBuildDialogBox%.
Description	Sets the Dialog Box Title Bar title offset.
Usage	**bwDB(0).titlebaroffset =** ***setting%***

Remarks

The Title Text in a Dialog Box Title Bar may be placed any where in the bar. To offset the title from the left edge of the title bar set the offset value, in pixels, in this property. By default this property is set to zero (0), this will cause the title text to be centered in the title bar. **Note:** Be careful not to set the offset value to high or the title text will be displayed outside of the dialog box.

setting%	Integer value indicating the dialog box title bar offset in pixels.

Data Type Integer

ulcase *Property*

Applies To gComboBox% and gEditBox%.

Description Sets a flag which indicates that an Edit Box Control will convert it's entry to upper case.

Usage **bwEC(0).ulcase =** ***setting%***

Remarks Set this property to True (-1) to make the Edit Box Control convert it's entry to upper case. By default this flag is set to False (0).

setting% Integer value indicating the state of the flag.

True (-1) = Active
False (0) = Inactive

Data Type Integer

update *Property*

Applies To	gBuildDialogBox%, gPullDownMenu%, gCommandButton%, gEditBox%, gListBox%, gChkOptBox%, gHorzScrollBar%, gVertScrollBar% and gComboBox%.
Description	Sets a flag which indicates that a Control Object should perform an update event.
Usage	**bwLB(0).update =** ***setting%*** ***(shown as List Box)***
Remarks	When this property is set to True (-1) a Control Object performs an update event, which draws the control's action to the display. This event usually happens when a Control Object is registered. Note that it is usually not necessary to have a paint event when an update event is requested. An **update** event differs from a **paint** event in that the paint event will paint the entire control (i.e. its border, scroll bars, etc.) and an update event will only paint the action of the control. For example an update of a List Box will only paint the items in the list and not the border or scroll bars.

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

See Also	**addobj, objidnum, paintobj**
Data Type	Integer

value *Property*

Applies To gHorzScrollBar% and gVertScrollBar%.

Description Sets and returns the current state of the control.

Usage **bwSB(0).value =** ***setting!***

Remarks The value determines the current position of the scroll bar button, which is always an amount between the **min** and **max** property values.

setting!	Single Integer value indicating the position of the scroll bar button.

See Also **min, max, smallchange, largechange**

Data Type Single

ws3dflag *Property*

Applies To	gBuildDialogBox%.
Description	Sets a flag which indicates that the Dialog Box background will be displayed as a 3d bar.
Usage	**bwDB(0).ws3dflag =** ***setting%***
Remarks	Set this property to True (-1) to have the Dialog Box background displayed as a 3d bar. The background color is also forced to the color gray. By default this property flag is set to False (0).

setting%	Integer value indicating the state of the flag. True (-1) = Active False (0) = Inactive

Data Type	Integer

x1 *Property*

Applies To	gCommandButton%, gChkOptBox%,gHorzScrollBar%, gVertScrollBar% and gListBox%.
Description	Sets and returns the x position of the upper left corner of a control.
Usage	**bwLB(0).x1 =** ***setting%*** ***(shown as List Box)***
Remarks	This property will set the x value of the upper left corner of a control. The value is set in pixels and should be offset from the upper left corner of the dialog box it resides in.
	setting% Integer value indicating the x location of the upper left corner of the control in pixels.
See Also	**y1**
Data Type	Integer

y1 *Property*

Applies To gCommandButton%, gChkOptBox%,gHorzScrollBar%, gVertScrollBar% and gListBox%.

Description Sets and returns the y position of the upper left corner of a control.

Usage **bwLB(0).y1 =** ***setting%***
(shown as List Box)

Remarks This property will set the y value of the upper left corner of a control. The value is set in pixels and should be offset from the upper left corner of the dialog box it resides in.

setting%	Integer value indicating the y location of the upper left corner of the control in pixels.

See Also **x1**

Data Type Integer

Unsupported *Property*

The following properties are listed in the EGUI include files but are **NOT** supported in this version of the EGUI Library.

arraysz
dbfileflag
fileflag
hibreak
justify *
keypress
listfile
lobreak
mb
menulinehgt *
menustatuscolr
mstatus *
mx
my
nontitleflag
numofrecs
parx1
pary1
parx2
pary2
recstart
reserve1
sizeflag
titlebarwid

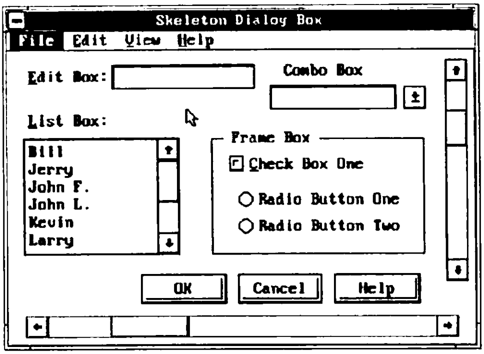
* These items are supported but are for internal system use only.

Control Functions

Chapter 12

gBuildDialogBox% *Control Function*

This procedure builds a Dialog Box Window for placing other Controls on. You should use this procedure when you wish to build a Dialog Box. There are several procedures which are required to build a Dialog Box frame, this procure will greatly ease this process. **Important:** this procedure must be called prior to painting any other controls. See *Window Types* for more information on Dialog Box Windows. Below is a typical Dialog Box:



Property Prefix **bwDB(0).** All properties must be preceded by there prefix for proper operation. (*Example:* **bwDB(0).paint=0)**

Properties

paintobj	addobj	objidnum *
update *	nodreset *	enable
objid *	status *	dbx1
dby1	dbx2	dby2
dclor	nonmoveflag	noncloseflag
nontitleflag *	nonborderflag	sizeflag *
ws3dflag	sysfontnum	titlebarwid
title	titlebaroffset	

* *These properties are reserved for future development and are not currently supported.*

Action

Draws a Dialog Box Window using the parameters set in the Dialog Box Properties.

Syntax

gBuildDialogBox%

Remarks

Always use this procedure for creating a new Dialog Box Window, because it automates the process of creating a Dialog Box Window by combining the standard procedures used to create a new window.

The Dialog Box Properties must be set prior to calling this procedure. The properties and an example of how use this procedure is shown below.

Note: The Pull Down Menu property **pdmenuflag** is not part of the Dialog Box structure. This property is used solely by the EGUI Form Generator.

See Also

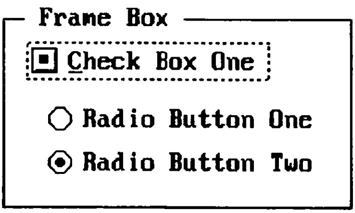
gRemoveDialogBox%

Example

```
'This Code should be located in the Draw Dialog Box Section
'of the DBFormat
'Build Dialog Box ---------------------------------------
 GOSUB gSkeltonDBReSetDialogBox       'Set Properties
 retcode% = gBuildDialogBox%
  .
  .
  .

'Place this code in the Dialog Box Properties Section of the
'DBFormat
 gSkeltonDBReSetDialogBox:     'Set Properties

      bwDB(0)paintobj = 0        'Always Zero
      bwDB(0)addobj = 0          'Always Zero
      bwDB(0)enable = -1         'Ignored for DB
      bwDB(0)dbx1 = x1%          'Current Static Coords
      bwDB(0)dby1 = y1%          'Current Static Coords
      bwDB(0)dbx2 = x2%          'Current Static Coords
      bwDB(0)dby2 = y2%          'Current Static Coords
      bwDB(0)nonmoveflag = 0     'Movalbe Box
      bwDB(0)noncloseflag = 0    'Close Button Enabled
      bwDB(0)nontitleflag = 0    'Title Bare Exists
      bwDB(0)nonborderflag = 0   'Border Enabled
      bwDB(0)ws3dflag = 0        'Standard Background
      bwDB(0)title = "Skeleton Dialog Box"

 RETURN
```

gChkOptBox% *Control Function*

The Check Option Box Control displays a check button or a radio button with text to the right side of the button. A *Check Button* is and option that can be turned on or off. Use this control to give the user a true/flase or yes/no option. Check Boxes may also be used in groups to display multiple choices from which the user may chose one or more options. A *Radio Button* is and option that can be turned on or off. Radio Buttons should be used as option buttons only. The difference between a check button and a radio button is that if a radio button is selected all other radio buttons should be canceled (turned off), whereas any check buttons may be on or off. Below is a typical check button and two radio buttons.



Property Prefix	**bwBT(0).**	All properties must be preceded by there prefix for proper operation. (*Example:* **bwBT(0).paint=0)**

Properties

paintobj
update
enable
x1
buttontype
border
accelkey

addobj
nodreset *
objid
y1
dcolr
sysfontnum
button **

objidnum *
tabflag
status
actsusflag
active
shadowflag

* *These properties are reserved for future development and are not supported.*
** *These properties are not supported in the EGUI Generator, only the library.*

Action Draws a Check Box or Radio Button using the parameters set with Check Option Box Properties.

Syntax **gChkOptBox%***(text$)*

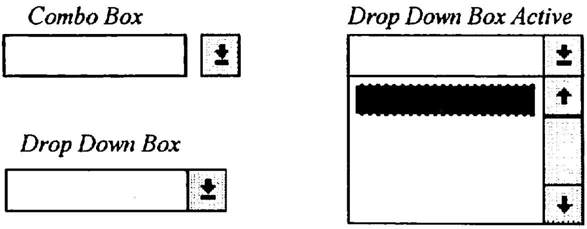
Remarks Use the **bwBT(0).button** property to select a Check Box or Option Button. **Note:** Option Buttons are grouped in the Generator <u>only</u>.

Example

```
'This code should be located in the Draw Dialog Box Section
'of the DBFormat
'Setup Draw Chk Box Number 0 ----------------------------
 GOSUB EXAMPLEReSetChkBox0
 bwBT(0).paintobj = -1                 'Paint the Object
 bwBT(0).addobj = -1                   'Add Obj to Object
       List
 bwBT(0).update = -1                   'Update the Object
 retcode% = gChkOptBox%(text$)

'Place this code in a Case Statement in the Window Main Loop
 CASE case_select_number
     GOSUB EXAMPLEReSetChkBox0         'Set Obj Properties
     retcode% = gChkOptBox%(Text$)     'Call the Ctrl Obj
     ChkBox0flag% = bwBT(0).active     'Reset Local Poniter
     IF retcode% = -3 THEN
        GOSUB EXAMPLEProcessKey        'Process Key Press if
        any
     END IF

'Place this code in the Dialog Box Properties Section of the
'DBFormat
 EXAMPLEReSetChkBox0:                  'Set Local Properties
     bwBT(0).paintobj = 0
     bwBT(0).addobj = 0
     bwBT(0).update = 0
     bwBT(0).tabflag = -1
     bwBT(0).enable = -1
     bwBT(0).objid = -1
     bwBT(0).status = 0
     bwBT(0).x1 = (x1% + 40)
     bwBT(0).y1 = (y1% + 112)
     bwBT(0).actsusflag = 0
     bwBT(0).depress = 0
     bwBT(0).objheight = bwCheckButtonWid%    'Set Std But Hgt
     bwBT(0).objwidth = bwCheckButtonWid%     'Set Std But Wid
     bwBT(0).buttontype = 1
     bwBT(0).dcolr = 0
     bwBT(0).active = ChkBox0flag%        'Pass Local Pointer
     bwBT(0).border = 0
     bwBT(0).sysfontnum = 1
     bwBT(0).shadowflag = 0
     bwBT(0).button = 0             '0=CheckBox 1=OptionButton
     bwBT(0).accelkey = 0
     Text$ = "Check Box 0"
 RETURN
```

gComboBox% *Control Function*

A *Combo Box Control* combines the features of an *Edit Box* and a *List Box.* Use this control to enable the user to make a selection by typing text into a edit box or by selecting an item from the list below it. Below are three typical combo boxes.



Property Prefix **bwEC(0).** All properties must be preceded by there prefix for proper operation. (*Example:* **bwEC(0).paint=0)**

Properties

paintobj
update
enable
combo
col
dlen
autotab
rate
border
sysfontnum
num2disp
addobj
nodreset *
objid
dropdown
noeditflag
oiflag
eptr **
ulcase
sortflag
shadowflag
novsbflag
objidnum *
tabflag
status
row
slen
dcolr
dptr **
format
movetoflag
numflag
hsbflag

* *These properties are reserved for future development and are not supported.*
** *These properties are not supported in the EGUI Generator, only the library.*

Action Draws a Combo Box using the parameters set with the Combo Box Properties.

Syntax **gComboBox%***(comboselection$, combolist$(), reserved%)*

Remarks The properties **bwEC(0).combo** and **bwEC(0).dropdown** select whether the combo box is a drop down type or a combo type. A drop down box does not allow the user to type text in the edit box, instead the item selected from the list is displayed in the edit box.

NOTE: The Edit Box Control must precede the Combo Box Control in the **DBFormat Main Loop** Case Statement for proper operation of a Combo Box. The Current Object Number is managed internally by the Object Manager during tabs and object selection.

NOTE: To Enable or Disable a Combo Box the **bwEC(0).paintobj** and **bwEC(0).addobj** properties must be True when setting the **bwEC(0).enable** to True or False. Also **bwEC(0).slen** and **bwEC(0).dlen** must have the same values as when originally added to the *Object Control List.*

Example

```
'This code should be located in the Draw Dialog Box Section
'of the DBFormat
'Setup Combo Box Edit Box Number 2
             ----------------------------
 GOSUB EXAMPLEReSetComboBox2
 bwEC(0).paintobj = -1
 bwEC(0).addobj = -1
 bwEC(0).update = -1
 retcode% = gEditBox%(ComboSelection2$)

'Setup Combo Box Button Number 2
             ------------------------------
 GOSUB EXAMPLEReSetComboBox2
 bwEC(0).paintobj = -1
 bwEC(0).addobj = -1
 bwEC(0).update = -1
 retcode% = gComboBox%(ComboSelection2$, ComboList2$(),_
        Reserved%)
```

(Continued On Next Page)

```
'Place this code in a Case Statement in the Window Main Loop
 CASE case_select_number
     GOSUB EXAMPLEReSetComboBox2
     retcode% = gEditBox%(ComboSelection2$)
     IF retcode% = -3 THEN
         GOSUB EXAMPLEProcessKey
     END IF

 CASE case_select_number
     GOSUB EXAMPLEReSetComboBox2
     retcode% = gComboBox%(ComboSelection2$, ComboList2$(),_
                                              Reserved%)
     IF retcode% = -3 THEN
         GOSUB EXAMPLEProcessKey
     END IF


'Place this code in the Dialog Box Properties Section of the
'DBFormat

 EXAMPLEReSetComboBox2:
     bwEC(0).paintobj = 0
     bwEC(0).addobj = 0
     bwEC(0).update = 0
     bwEC(0).tabflag = -1
     bwEC(0).enable = -1
     bwEC(0).objid = -1
     bwEC(0).status = 0
     bwEC(0).dcolr = 0
     bwEC(0).sysfontnum = 0
     bwEC(0).combo = -1
     bwEC(0).dropdown = 0
     bwEC(0).row = (y1% +56)
     bwEC(0).col = ((x1% \ 8) + 34)
     bwEC(0).noeditflag = 0
     bwEC(0).slen = 255
     bwEC(0).dlen = 15
     bwEC(0).oiflag = 0
     bwEC(0).autotab = 0
     bwEC(0).eptr = 0                    'For Internal Use
     bwEC(0).dptr = 1                    'For Internal Use
     bwEC(0).rate = 4
     bwEC(0).ulcase = 0
     bwEC(0).format = 0
     bwEC(0).justify = 0                 'For Internal Use
     bwEC(0).border = -1
     bwEC(0).shadowflag = -1
     bwEC(0).numflag = 0
     bwEC(0).sortflag = 0
     bwEC(0).movetoflag = 0
     bwEC(0).num2disp = 6
     bwEC(0).novsbflag = 0
     bwEC(0).hsbflag = 0
     bwEC(0).hpagesz = 1
     bwEC(0).mx = 0                      'For Internal Use
     bwEC(0).my = 0                      'For Internal Use
     bwEC(0).mb = 0                      'For Internal Use
     bwEC(0).keypress = 0                'For Internal Use
     Text$ = "ComboBox2"
 RETURN
```

gCommandButton% *Control Function*

A *Command Button Control* should be used to perform a task when selected by the user, who either clicks the button with the left mouse button or presses the ENTER key when the button has the focus.

Below are some typical command buttons:

Property Prefix	**bwBT(0).**	All properties must be preceded by there prefix for proper operation. (*Example:* **bwBT(0).paint=0)**

Properties

paintobj	addobj	objidnum *
update	nodreset *	tabflag
enable	objid	status
x1	y1	depress **
objheight	objwidth	buttontype
dcolr	border	sysfontnum
assignobj	accelkey	icon
iconx1off	icony1off	logicflag
iconfile		

* *These properties are reserved for future development and are not supported.*
** *These properties are not supported in the EGUI Generator, only the library.*

Action Draws a Command Button using the parameters set with the Command Button Properties.

Syntax **gCommandButton%***(text$)*

Remarks To display text on a *Command Button*, assign the text information to the text$ property.

To display and *Icon* on a *Command Button*, set the **bwBT(0).icon** property to **True** (-1) and assign an Icon Filename, with optional drive and path if desired, to the **bwBT(0).iconfile** property *(Note: It is recommended to use the IconPath$ environment variable for assigning the icon path info).* You may position the icon on the button with the **bwBT(0).iconx1off** and **bwBT(0).icony1off** properties.

Example

```
'This code should be located in the Draw Dialog Box Section
'of the DBFormat
'Setup Button Number 1
         ---------------------------------------
 GOSUB EXAMPLEReSetButtonl
 bwBT(0).paintobj = -1
 bwBT(0).addobj = -1
 bwBT(0).update = -1
 retcode% = gCommandButton%(text$)


'Place this code in a Case Statement in the Window Main Loop
CASE case_select_number

    GOSUB EXAMPLEReSetButtonl
    retcode% = gCommandButton%(Text$)
    IF retcode% = -2 THEN

        'Place Task to perform here

    ELSEIF retcode% = -3 THEN
        GOSUB EXAMPLEProcessKey
    END IF
```

(Continued On the Next Page)

```
'Place this code in the Dialog Box Properties Section of the
'DBFormat

 EXAMPLEReSetButton1:
     bwBT(0).paintobj = 0
     bwBT(0).addobj = 0
     bwBT(0).update = 0
     bwBT(0).tabflag = -1
     bwBT(0).enable = -1
     bwBT(0).objid = -1
     bwBT(0).status = 0
     bwBT(0).x1 = (x1% + 32)
     bwBT(0).y1 = (y1% + 232)
     bwBT(0).depress = 0
     bwBT(0).objheight = 24
     bwBT(0).objwidth = 96
     bwBT(0).buttontype = 2
     bwBT(0).dcolr = 0
     bwBT(0).border = -1
     bwBT(0).sysfontnum = 1
     bwBT(0).assignobj = 0
     bwBT(0).accelkey = 0
     bwBT(0).icon = 0
     bwBT(0).iconxloff = 5
     bwBT(0).iconyloff = 5
     bwBT(0).logicflag = 0
     bwBT(0).iconfile = ""
     Text$ = "Button1"
 RETURN
```

gEditBox% *Control Function*

An *Edit Box Control* displays information you specify or the user enters. The *Edit Box* will only display one line of text to be edited, for multiple lines use a *List Box Control* with the **bwLB(0).editflag** property set to **True** (-1).

Below is a typical Edit Box, the text to the left of the box is displayed using **gDrawTextCol%**.

Edit Box: Text

Property Prefix	**bwEC(0).**	All properties must be preceded by there prefix for proper operation. (*Example:* **bwEC(0).paint=0**)

Properties

- paintobj
- update
- enable
- row
- slen
- dcolr
- dptr **
- format
- shadowflag
- addobj
- nodreset *
- objid
- col
- dlen
- autotab
- rate
- border
- numflag
- objidnum *
- tabflag
- status
- noeditflag
- oiflag
- eptr **
- ulcase
- sysfontnum

* *These properties are reserved for future development and are not supported.*
** *These properties are not supported in the EGUI Generator, only the library.*

Action Draws a Edit Box using the parameters set with Edit Box Properties.

Syntax **gEditBox%***(text$)*

Remarks Only charcter strings may be edited in an Edit Box. To edit numeric input, use the **STR$** Function to convert the value to a string before passing it to the Edit Box Control. After editing the value you may convert it back to a numeric type with the **VAL** Function.

Example

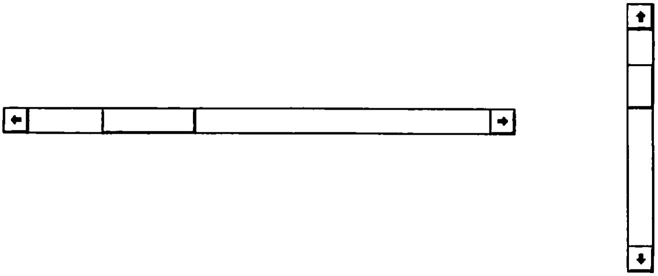
```
'This code should be located in the Draw Dialog Box Section
'of the DBFormat
'Setup Edit Box Number 1
         ------------------------------------
 GOSUB EXAMPLEReSetEditBox1
 bwEC(0).paintobj = -1
 bwEC(0).addobj = -1
 bwEC(0).update = -1
 retcode% = gEditBox%(Text$)

'Place this code in a Case Statement in the Window Main Loop
CASE case_select_number

     GOSUB EXAMPLEReSetEditBox1
     retcode% = gEditBox%(Text$)
     IF retcode% = -3 THEN
        GOSUB EXAMPLEProcessKey
     END IF
```

(continued of next page)

```
'Place this code in the Dialog Box Properties Section of the
'DBFormat

 EXAMPLEReSetEditBox1:
     bwEC(0).paintobj = 0
     bwEC(0).addobj = 0
     bwEC(0).update = 0
     bwEC(0).tabflag = -1
     bwEC(0).enable = -1
     bwEC(0).objid = -1
     bwEC(0).status = 0
     bwEC(0).dcolr = 0
     bwEC(0).sysfontnum = 0
     bwEC(0).row = (y1% +56)
     bwEC(0).col = ((x1% \ 8) + 11)
     bwEC(0).noeditflag = 0
     bwEC(0).slen = 255
     bwEC(0).dlen = 10
     bwEC(0).oiflag = 0
     bwEC(0).autotab = 0
     bwEC(0).eptr = 0                     'For Internal Use
     bwEC(0).dptr = 1                     'For Internal Use
     bwEC(0).rate = 4
     bwEC(0).ulcase = 0
     bwEC(0).format = 0
     bwEC(0).justify = 0                  'For Internal Use
     bwEC(0).border = -1
     bwEC(0).shadowflag = -1
     bwEC(0).numflag = 0
     bwEC(0).mx = 0                       'For Internal Use
     bwEC(0).my = 0                       'For Internal Use
     bwEC(0).mb = 0                       'For Internal Use
     bwEC(0).keypress = 0                 'For Internal Use
     Text$ = "EditBox1"
 RETURN
```

gHorzScrollBar% & gVertScrollBar% *Control Function*

Scroll Bars are graphical tools for quickly navigating through a long list of items or a large amount of information, and for indicating the current position on a scale. A scroll bar can also be used as an input device or an indicator of speed or quantity; for example, to control the adjustment of a custom color or to view the time elapse in a timed process. There are two scroll bar functions, Horizontal and Vertical. Both use the same properties but have different function names. Below are two typical scroll bars.



Property Prefix	**bwSB(0).**	All properties must be preceded by there prefix for proper operation. (*Example:* **bwSB(0).paint=0)**

Properties

paintobj	addobj	objidnum *
update	nodreset *	tabflag
enable	objid	status
x1	y1	objheight
objwidth	frame	value
min	max	smallchange
largechange		

* *These properties are reserved for future development and are no supported.*

Action Draws a Horizontal or Vertical Scroll Bar using the parameters set with Scroll Bar Properties.

Syntax **gHorzScrollBar%**
gVertScrollBar%

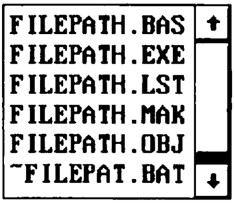
Remarks A Horizontal Scroll Bar is used for the example, however to create a Vertical Scroll Bar replace **gHorzScrollBar%** with **gVertScrollBar%** and set the properties **bwSB(0).objheight** to the height in pixels and **bwSB(0).objwidth** = **bwStandButwid%.**

Example

```
'This code should be located in the Draw Dialog Box Section
'of the DBFormat
'Setup Horz Scroll Bar Number 1 ----------------------------

 GOSUB EXAMPLEReSetHorzSBar1
 bwSB(0).paintobj = -1
 bwSB(0).addobj = -1
 bwSB(0).update = -1
 retcode% = gHorzScrollBar%

'Place this code in a Case Statement in the Window Main Loop
 CASE case_select_number
     GOSUB EXAMPLEReSetHorzSBar1
     retcode% = gHorzScrollBar%
     bwHorzSBarVal1! = bwSB(0).value
     IF retcode% = -3 THEN
         GOSUB EXAMPLEProcessKey
     END IF

'Place this code in the Dialog Box Properties Section of the
'DBFormat
 EXAMPLEReSetHorzSBar1:
     bwSB(0).paintobj = 0
     bwSB(0).addobj = 0
     bwSB(0).update = 0
     bwSB(0).tabflag = -1
     bwSB(0).enable = -1
     bwSB(0).objid = -1
     bwSB(0).status = 0
     bwSB(0).x1 = (x1% + 24)
     bwSB(0).y1 = (y1% + 288)
     bwSB(0).objheight = bwStandButwid%
     bwSB(0).objwidth = 400
     bwSB(0).frame = 20
     bwSB(0).value = bwHorzSBarVal1!
     bwSB(0).min = 0
     bwSB(0).max = 100
     bwSB(0).smallchange = 2
     bwSB(0).largechange = 20
 RETURN
```

gListBox% *Control Function*

A *List Box* displays a list of items from which the user can choose one; an exception to this is when the **bwLB(0).tagflag** property is set to **True** the user may select (tag) as many items as desired. A vertical scroll bar is automatically added to the list box to allow scrolling of items. Below is a typical list box.



```
FILEPATH.BAS
FILEPATH.EXE
FILEPATH.LST
FILEPATH.MAK
FILEPATH.OBJ
~FILEPAT.BAT
```

Property Prefix	**bwLB(0).**	All properties must be preceded by there prefix for proper operation. (*Example:* **bwLB(0).paint=0**)

Properties

paintobj	addobj	objidnum *
update	nodreset *	tabflag
enable	objid	status
xl	yl	dptr **
aptr **	rptr **	dcolr
tagflag	novsbflag	hsbflag
hpagesz	elmwid **	charwid
chrhgt	intflag **	noborderflag
headerflag *	headercolr *	headerfont *
editflag	movetoflag	dblclkflag
clearflag	hlineflag *	hlinecolr *
sortflag	sysfontnum	shadowflag

* *These properties are reserved for future development and are not supported.*

** *These properties are not supported in the EGUI Generator, only the library.*

Action Draws a List Box using the parameters set with the List Box Properties.

Syntax **gListBox%***(array$(), arrayheader$)*

Remarks No remarks.

Example

```
'This code should be located in the Draw Dialog Box Section
'of the DBFormat
'Setup List Box Number 1 ----------------------------------

 GOSUB EXAMPLEReSetListBox1
 bwLB(0).paintobj = -1
 bwLB(0).addobj = -1
 bwLB(0).update = -1
 retcode% = gListBox%(Array1$(), ArrayHeader1$)

'Place this code in a Case Statement in the Window Main Loop
 CASE case_select_number
     GOSUB EXAMPLEReSetListBox1
     retcode% = gListBox%(Array1$(), ArrayHeader1$)
     IF retcode% = -3 THEN
         GOSUB EXAMPLEProcessKey
     END IF
```

(continued on next page)

```
'Place this code in the Dialog Box Properties Section of the
'DBFormat

 EXAMPLEReSetListBox1:
     bwLB(0).paintobj = 0
     bwLB(0).addobj = 0
     bwLB(0).update = 0
     bwLB(0).tabflag = -1
     bwLB(0).enable = -1
     bwLB(0).objid = -1
     bwLB(0).status = 0
     bwLB(0).x1 = (x1% + 280)
     bwLB(0).y1 = (y1% + 123)
     bwLB(0).dptr = 1                        'For Internal Use
     bwLB(0).aptr = 1                        'For Internal Use
     bwLB(0).rptr = 1                        'For Internal Use
     bwLB(0).tagflag = 0
     bwLB(0).arraysz = -1                    'For Internal Use
     bwLB(0).novsbflag = 0
     bwLB(0).hsbflag = 0
     bwLB(0).hpagesz = 1                     'Lib Supported Only
     bwLB(0).elmwid = 14                     'Lib Supported Only
     bwLB(0).charwid = 14
     bwLB(0).charhgt = 8
     bwLB(0).intflag = -256                  'For Internal Use
     bwLB(0).headerflag = 0                  'Lib Supported Only
     bwLB(0).headercolr = 0                  'Lib Supported Only
     bwLB(0).editflag = 0
     bwLB(0).movetoflag = 0
     bwLB(0).dblclkflag = 0
     bwLB(0).clearflag = 0
     bwLB(0).sortflag = 0
     bwLB(0).sysfontnum = 0
     bwLB(0).shadowflag = -1
 RETURN
```

gObjManager% *Control Function*

The EGUI System is based on the usage of Program Objects linked together to build your applications User Interface. The *Object Manager* does the task of keeping all the objects organized and controllable. Use the Object Manager to add, remove, find or modify any object which have been created. The Object Manager is not accessible through the EGUI Generator only the EGUI Library when you are writing code. Most of the code necessary to use the Object Manager will be created when you build a Function or Module from the Generator.

Property Prefix	**bwOL(0).**	All properties must be preceded by there prefix for proper operation. *Example:* **bwOL(0).addobjflag=-1)**

Properties

objx1	objy1	objx2
objy2	reobjx1	reobjx2
reobjy2	tabflag	enable
addobjflag	objidnum	

NOTE: None of these properties are accessible through the EGUI Generator only the EGUI Library.

Action Manage all objects which exists in the EGUI System.

Syntax **gObjManager%**

Remarks The most important property of the object manager is the **bwOL(0).addobjflag**. This property is used to select the action you want the object manager to take on an object. Below is a list of those options.

-1 = Add Object to Object List
-2 = Search for an Object in the List
-3 = Remove Object from the List
-4 = Get Next Active Object
-5 = Get Last Active Object
-6 = Set Active Object Boundaries
-7 = Set Dialog Box Move Object
-8 = Enable and Object
-9 = Disable and Object
-10 = Reassign Object Perimeter
-11 = Set Button Assignment Obj Number
-12 = Get Object Status

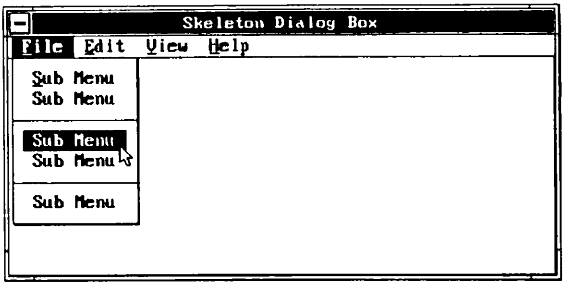
Note: When using option -12 the **bwOL(0).objidum** property must be set with the Object Handle you wish to get prior to calling. The object handle can be obtained when you create the object from the **bwOL(0).objidnum** property.

Example

```
'Add Object to Object List
 -------------------------------------
  bwOL(0).objx1 = 100        'Set the Object Rectangle Area
  bwOL(0).objy1 = 100
  bwOL(0).objx2 = 200
  bwOL(0).objy2 = 300
  bwOL(0).enable = -1        'Enable the Object
  bwOL(0).tabflag = -1       'Enable Tab Process
  bwOL(0).addobjflag = -1    'Add the Object to the List
  retcode% = gObjManager%    'Call the Object Manager
```

gPullDownMenu% *Control Function*

A *Pull Down Menu* displays a customized menu for your application. Items on a menu can include commands the user can choose to carry out an action, submenu names, and separator bars. **Note***: a menu should not exceed the Dialog Box borders when pulled down, so adjust your menu items accordingly.* Below is a typical Pull Down Menu:



Property Prefix	**bwPD(0).**	All properties must be preceded by there prefix for proper operation. *Example:* **bwPD(0).addobjflag=-1)**

Properties

paintobj	addobj	objidnum *
update	nodreset *	enable
objid	status	parx1
pary1	parx2	pary2
menx1	meny1	menx2
meny2	menucolr	menutextcolr
menustatuscolr	menulinehgt	sysfontnum
shadowflag		

** These properties are reserved for future development and are not currently supported.*

NOTE: None of these properties are accessible through the EGUI Generator only the EGUI Library.

Action Draw a *Pull Down Menu* using the parameters set with Pull Down Menu Properties.

Syntax

gPullDownMenu%*(menuitem$(),menustatus%(),menuacckey%(),*

_

barmenu%,pdmenu%,keypress%)

Remarks **Important:** Before calling the *Pull Down Menu* object you must first set the object up using **gSetPDMenu%** in the *Draw Dialog Box Section* of the **DBFormat.** It is also important to note that the **pdmenuflag** property, *(located within the Dialog Box properties)*, in the EGUI Generator is used to turn a menu **On** and **Off** in the Generator, and this property is not accessible in the Library.

menuitems$ Variable length string array which holds the items in a menu list. This array must be dimensioned to the correct maximum menu size prior to assigning items. By default the EGUI Generator will set the maximum menu size to **15**. This may be increased or decreased in code as needed.

menustatus% Integer array which indicates if a menu item is unselectable or selectable. To make the item unselectable *(grayed out)* set this value to True (-1). Else a False (0) value makes the item selectable.

menuacckey% Integer array which indicates if a menu item has an Accelerator Key and if so which character. Set to zero for no Accelerator otherwise set to the character number in the item.

barmenu% Integer value which returns the bar menu selected.

pdmenu% Integer value which returns the pull down menu item selected.

keypress%	Integer value which returns the value of the key which was pressed.

(continued on next page)

NOTE: The function returns a -3 if a menu selection is made.

The arrays ***menuitemS(), menustatus%(),*** and ***menuacckey%()*** must be dimensioned to there proper size before calling **gPullDownMenu%**. The variable *NumofSelection%* is used to dimension a multidimensional array for the pull down list. This variable should be adjusted to a size large enough to accommodate the largest list, however make sure that when the list is pulled down it does not extend past the Dialog Box borders. The other variable used to dimension the item array is *NumofMenus%* which sets the total number of pull down menu lists.

See Also **gSetPDMneu%**

Example

See the following pages for sample code for building a Dialog Box with a Pull Down Menu.

Example *(continued)*

```
'Egui Application Function Module Pull Down Menu Example
'
'    AUTHOR: Your Name
'
'    FileName: F:\EGUI\PDMENU.BAS
'
'

   'Load Include Files
              ------------------------------------------
    REM $INCLUDE: 'BWENV.INC'
    REM $INCLUDE: 'BWPRP.INC'
    REM $INCLUDE: 'BWCTR.INC'


   'Declare Internal Basic Functions
              ------------------------------
    DECLARE FUNCTION PDMTEST% ()
    DECLARE FUNCTION PDMTESTReSetPDMenuProps% ()

   'Setup Error Handling [Optional]
              ------------------------------
    ON ERROR GOTO MainExit                  'Optinal Code

   'Clear Extra Stack Space
              -------------------------------------
    CLEAR , , 3000

   'Set Standard Screen Mode 12 VGA (640x$80 16 Color )
              -------------
    Vmode% = 12
    ClrScrflag% = -1
    retcode% = gSetVideoMode%(Vmode%, ClrScrflag%)

   'Initialize Egui System
              --------------------------------------
    IF NOT retcode% THEN
        retcode% = gInitBWSystem%
    ELSE
        GOTO MainExit
    END IF

   'Main Function Call
              ------------------------------------------
    retcode% = PDMTEST%


MainExit:

   'Hide Cursor & Reset Screen Mode to Text
              -----------------------
    retcode% = gHideMouse%
    Vmode% = 0
    ClrScrflag% = -1
    retcode% = gSetVideoMode%(Vmode%, ClrScrflag%)

   'If Error Occured Print Error Code and Message on Exit[Optional]
```

```
    IF ERR THEN
        PRINT "Error "; 0; Occured, Program; Aborted.; ""
    END IF

   'End Program -------------------------------------------------
    END

  '

FUNCTION PDMTEST%

   'Setup Error Handler -----------------------------------------
    ON LOCAL ERROR GOTO PDMTESTError

   'Setup Dialog Box Boundery Parameters -------------------------
    IF x2% = 0 THEN
       x1% = 96: y1% = 136
       x2% = 544: y2% = 400
    END IF

   'Setup Pull Down Menu Arrays ----------------------------------
    NumofPDMenus% = 2       'Number of Menu Bar Items
    NumofSelections% = 15  'Number of Selections for each Pull Down
    REDIM MenuItems$(NumofPDMenus%, NumofSelections%)
    REDIM MenuStatus%(NumofPDMenus%, NumofSelections%)
    REDIM MenuAccKey%(NumofPDMenus%, NumofSelections%)
    REDIM mWinbar%(NumofPDMenus%, 2)
    REDIM mWinmen%(NumofPDMenus%, NumofSelections%, 2)
    REDIM menbarpos%(UBOUND(MenuItems$))
    REDIM menselpos%(UBOUND(MenuItems$), 2)

   'Build Dialog Box --------------------------------------------
    GOSUB PDMTESTBuildDialogBox

   'Setup Main Loop ---------------------------------------------

        DO

            SELECT CASE CurrentObjNum%

            CASE -1                                  'CASE -1 Close Dialog
                Box
                PDMTEST% = -1                        'Close Button
                Procedure
                EXIT DO

            CASE 0                                   'CASE 0 Move Dialog
                Box
                SavResflag% = -1
                retcode% = gScreenFiler%(Bx1%, By1%, Bx2%, By2%,_

                SavResflag%)
                x1% = movx1%: y1% = movy1%
                x2% = movx2%: y2% = movy2%
                GOSUB PDMTESTBuildDialogBox

            CASE 1
                retcode% = PDMTESTReSetPDMenuProps%
                retcode% = gPullDownMenu%(MenuItems$(),
                MenuStatus%(),_
                MenuAccKey%(), BarMenu%, PDMenu%, KeyPress%)
                IF retcode% = -3 THEN
                    GOSUB PDMTESTProcMenuSel
                    GOSUB PDMTESTProcessKey
                END IF

            CASE ELSE
                CurrentObjNum% = 1

            END SELECT

        LOOP



'---------------------------------------------------------------
PDMTESTExit:

   'Remove Dialog Box ---------------------------------------------
    retcode% = gRemoveDialogBox%

   'Exit Function -------------------------------------------------
    EXIT FUNCTION

RETURN

'---------------------------------------------------------------
PDMTESTBuildDialogBox:

   'Draw Dialog Box -----------------------------------------------
    GOSUB PDMTESTReSetDialogBox
    retcode% = gBuildDialogBox%

   'Setup Pull Down Menu ------------------------------------------
    retcode% = PDMTESTReSetPDMenuProps%
    bwPD(0).paintobj = -1
    bwPD(0).addobj = -1
    retcode% = gSetPDMenu%(MenuItems$(), MenuStatus%(),
                                                  MenuAccKey%())

   'Get Current Bounderies ----------------------------------------
    bwOL(0).addObjflag% = -6
    retcode% = gObjManager%

   'Set Current Object Number -------------------------------------
    CurrentObjNum% = 1

RETURN

'---------------------------------------------------------------
PDMTESTReSetDialogBox:

    bwDB(0).paintobj = 0
    bwDB(0).addobj = 0
    bwDB(0).enable = -1
    bwDB(0).dbx1 = x1%
    bwDB(0).dby1 = y1%
    bwDB(0).dbx2 = x2%
    bwDB(0).dby2 = y2%
    bwDB(0).nonmoveflag = 0
    bwDB(0).noncloseflag = 0
    bwDB(0).nonborderflag = 0
    bwDB(0).ws3dflag = 0
    bwDB(0).sysfontnum = 0
    bwDB(0).titlebaroffset = 0
    bwDB(0).title = 'Pull Down Menu Form'

RETURN

'-----------------------------------------------------------------
PDMTESTProcessKey:


RETURN


'-----------------------------------------------------------------
PDMTESTProcMenuSel:

   IF BarMenu% = 1 THEN

       IF PDMenu% = 1 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       ELSEIF PDMenu% = 2 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       ELSEIF PDMenu% = 3 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       ELSEIF PDMenu% = 5 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       ELSEIF PDMenu% = 7 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       END IF

   END IF

   IF BarMenu% = 2 THEN

       IF PDMenu% = 1 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       ELSEIF PDMenu% = 2 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       ELSEIF PDMenu% = 4 THEN

          'Place Code to Process on Pull Down Menu Selection Here!

       END IF

   END IF

RETURN
```

```
'----------------------------------------------------------------
PDMTESTError:

    PDMTEST% = ERR
    retcode% = gDispErrorMess%(ERR, ErrorMessage$, ErrorPostion$)
    RESUME PDMTESTExit

END FUNCTION


FUNCTION PDMTESTReSetPDMenuProps%

   'Setup Pull Down Menu Arrays ---------------------------------
    NumofPDMenus% = 2           'Number of Menu Bar Items
    NumofSelections% = 15       'Number of Selections for each Pull
               Down
    REDIM MenuItems$(NumofPDMenus%, NumofSelections%)
    REDIM MenuStatus%(NumofPDMenus%, NumofSelections%)
    REDIM MenuAccKey%(NumofPDMenus%, NumofSelections%)
    REDIM mWinbar%(NumofPDMenus%, 2)
    REDIM mWinmen%(NumofPDMenus%, NumofSelections%, 2)
    REDIM menbarpos%(UBOUND(MenuItems$))
    REDIM menselpos%(UBOUND(MenuItems$), 2)

   'Setup Pull Down Menu Control Properties --------------------
    bwPD(0).paintobj = 0
    bwPD(0).addobj = 0
    bwPD(0).update = 0
    bwPD(0).tabflag = -1
    bwPD(0).enable = -1
    bwPD(0).objid = 0
    bwPD(0).status = 0
    bwPD(0).menx1 = (bwDB(0).dbx1 + bwEV(0).Borderwid)
    bwPD(0).meny1 = (bwDB(0).dby1 + bwTitleBarHgt% + _
                      bwEV(0).Borderwid)
    bwPD(0).menx2 = (bwDB(0).dbx2 - bwEV(0).Borderwid)
    bwPD(0).meny2 = -1

   'Setup Menu Bar Items ---------------------------------------
    MenuItems$(1, 0) = "File"
    MenuStatus%(1, 0) = 0
    MenuAccKey%(1, 0) = 1
    MenuItems$(2, 0) = "Edit"
    MenuStatus%(2, 0) = 0
    MenuAccKey%(2, 0) = 1

   'Setup Menu Selection List 1 --------------------------------
    MenuItems$(1, 1) = "New"
    MenuStatus%(1, 1) = 0
    MenuAccKey%(1, 1) = 1
    MenuItems$(1, 2) = "Open"
    MenuStatus%(1, 2) = 0
    MenuAccKey%(1, 2) = 1
    MenuItems$(1, 3) = "Close"
    MenuStatus%(1, 3) = 0
    MenuAccKey%(1, 3) = 2
    MenuItems$(1, 4) = "-"
    MenuStatus%(1, 4) = 0
    MenuAccKey%(1, 4) = 0
    MenuItems$(1, 5) = "Print"
    MenuStatus%(1, 5) = -1
    MenuAccKey%(1, 5) = 1
    MenuItems$(1, 6) = "-"
    MenuStatus%(1, 6) = 0
    MenuAccKey%(1, 6) = 0
    MenuItems$(1, 7) = "Exit"
    MenuStatus%(1, 7) = 0
    MenuAccKey%(1, 7) = 2

   'Setup Menu Selection List 2 ----------------------------------
    MenuItems$(2, 1) = "Copy              Ctrl-C"
    MenuStatus%(2, 1) = 0
    MenuAccKey%(2, 1) = 1
    MenuItems$(2, 2) = "Paste             Ctrl-P"
    MenuStatus%(2, 2) = 0
    MenuAccKey%(2, 2) = 1
    MenuItems$(2, 3) = "-"
    MenuStatus%(2, 3) = 0
    MenuAccKey%(2, 3) = 0
    MenuItems$(2, 4) = "Delete"
    MenuStatus%(2, 4) = -1
    MenuAccKey%(2, 4) = 1

END FUNCTION
```

Library Functions

Chapter 13

g3DBox% *Function*

Action Draws a three dimensional filled box.

Syntax **g3DBox%***(x1%,y1%,xd%,yd%,invert%,border%,colr&)*

Remarks The following parameters are passed to the function:

x1%,y1%	Integer values indicating the top left corner of the box, in pixels.
xd%	Integer value indicating the width of the box in pixels.
yd%	Integer value indicating the height of the box in pixels.
invert%	If this parameter is set to -1 (True) the box will be displayed as an inverted 3D box.
border%	If this parameter set to -1 (True) a border will be drawn around the box using the system outline color (bwEV(0).Outline).
colr&	Long integer value indicating the background color attribute to paint the box, or -1 (True) for the box default color. *NOTE:* This is a single color value.

See Also **g3DRect%, gDrawRect%**

(see next page for example)

Example

```
x1% = 100
y1% = 100
xd% = 76
yd% = 28
invert % = 0
border% = -1
colr& = bwEV(0).Gray

retcode% = g3DBox%(x1%,y1%,xd%,yd%,invert%,border%,colr&)
```

g3DRect% *Function*

Action Draws a three dimensional rectangle outline.

Syntax **g3DRect%***(x1%,y1%,xd%,yd%,hcolr&,lcolr&)*

Remarks This procedure is used to create a three dimensional outline effect around a specific area of the screen. The following parameters are passed to the function:

x1%,y1% Integer values indicating the top left corner of the box, in pixels.

xd% Integer value indicating the width of the box in pixels.

yd% Integer value indicating the height of the box in pixels.

hcolr& Long integer value indicating the hightlite color attribute to paint the rectangles, or -1 (True) for the default color (bwEV(0).White).

lcolr& Long integer value indicating the lower color attribute to paint the rectangles, or -1 (True) for the default color (bwEV(0).DarkGray).

See Also **g3DBox%, gDrawRect%**

(see next page for example)

Example

```
x1% = 100
y1% = 100
xd% = 76
yd% = 28
hcolr& = -1             'Use Default Color
lcolr& = -1             'Use Default Color

retcode% = g3DRect%(x1%,y1%,xd%,yd%,hcolr&,lcolr&)
```

gBorder% *Function*

Action Draws a Dailog Box Border.

Syntax **gBorder%***(x1%,y1%,x2%,y2%,bwid%,colr&)*

Remarks This procedure is used to draw a border around a Dialog Box Window, however it may be used to draw a border around any area of the display. The following parameters are passed to the function:

x1%,y1%	Integer values indicating the top left corner of the border, in pixels.
x2%,y2%	Integer values indicating the bottom right corner of the border, in pixels.
bwid%	Integer value indicating the border width. If this value is greater than 0, then a Sizeable Border Style will be drawn. If this value is set to a negative number a border will be drawn with no size marks. **NOTE:** The **bwEV(0).Borderwid** value should be used to set this parameter if a sizeable border is to be drawn else set to zero.
colr&	Long integer value indicating the color attribute to paint the border. **NOTE:** The **bwEV(0).Bordercolr** or the **bwEV(0).InActBordercolr** should be used to set this parameter.

(see next page for example)

Example

```
x1% = 100
y1% = 100
x2% = 500
y2% = 350
bwid% = bwEV(0).Borderwid
colr& = bwEV(0).Bordercolr

retcode% = gBorder%(x1%,y1%,x2%,y2%,bwid%,colr&)
```

gCustomMouse% *Function*

Action Use this function to create a custom mouse cursor.

Syntax **gCustomMouse%** *(CurMask$, xHotSpot%, yHotSpot%)*

Remarks This procedure gives a method for building a custom mouse cursor when one of the system mouse cursors is not sufficient. The Cursor type for a custom mouse cursor should always equal 100. (**Note:** see the *Microsoft Mouse Programmer's Reference* for more information about the mouse.)

CurMask$	String indicating the pattern mask of the cursor. See the example below for how to build this mask.
xHotSpot%	The x offset for the mouse hotspot.
yHotSpot%	The y offset for the mouse hotspot.

See Also **gHideMouse%,gShowMouse%**

Note: See next page for example

Example

```
'Standard Mouse Mask Pattern

 curmask$ = ""
 curmask$ = curmask$ + "1111111111111111"
 curmask$ = curmask$ + "1001111111111111"
 curmask$ = curmask$ + "1000111111111111"
 curmask$ = curmask$ + "1000011111111111"
 curmask$ = curmask$ + "1000001111111111"
 curmask$ = curmask$ + "1000000111111111"
 curmask$ = curmask$ + "1000000011111111"
 curmask$ = curmask$ + "1000000001111111"
 curmask$ = curmask$ + "1000000000111111"
 curmask$ = curmask$ + "1000000000011111"
 curmask$ = curmask$ + "1000000000011111"
 curmask$ = curmask$ + "1000000011111111"
 curmask$ = curmask$ + "1001100001111111"
 curmask$ = curmask$ + "1111100001111111"
 curmask$ = curmask$ + "1111110000111111"
 curmask$ = curmask$ + "1111110000111111"

 curmask$ = curmask$ + "0000000000000000"
 curmask$ = curmask$ + "0000000000000000"
 curmask$ = curmask$ + "0010000000000000"
 curmask$ = curmask$ + "0011000000000000"
 curmask$ = curmask$ + "0011100000000000"
 curmask$ = curmask$ + "0011110000000000"
 curmask$ = curmask$ + "0011111000000000"
 curmask$ = curmask$ + "0011111100000000"
 curmask$ = curmask$ + "0011111110000000"
 curmask$ = curmask$ + "0011111111000000"
 curmask$ = curmask$ + "0011111000000000"
 curmask$ = curmask$ + "0010011000000000"
 curmask$ = curmask$ + "0000001100000000"
 curmask$ = curmask$ + "0000001100000000"
 curmask$ = curmask$ + "0000000110000000"
 curmask$ = curmask$ + "0000000000000000"

 xHotSpot% = 1
 yHotSpot% = 1

 gCustomMouse% (CurMask$, xHotSpot%, yHotSpot%)
```

gDispErrorMess% *Function*

Action

Displays a Window with an Error Message. This procedure is supplied to remove the need to build a new Dialog Box each time an Error Message needs to be displayed. If the information which may be displayed in this procedure does not meet your needs consider using the **gMessageDialog%** procedure or creating your on custom Dialog Box.

Syntax

gDispErroMess%*(errorcode%,errormessage$,errorpostion$*

Remarks

Use this procedure in the Local Error Handler to display any error messages which occur.

errorcode% Integer values indicating the Error Code which occurred, this may be a BASIC or DOS Error Code or a custom Error Code, but a distinction of which should be made in the Error Message String.

errormessage$ A short Error Message identifying the Error that occurred.

errorposition$ A short message pointing to the position that the error occurred.

Example:

errorposition$="MOD:Main-FUNC:gMain Win"

Example

```
errorcode% = 53
errormessage$ = 'DOS-File Not Found"
errorposition$='MOD:Main-FUNC:gMainWin'


          retcode%=gDispErroMess%(errorcode%,errormessage$,err
          orpostion$)
```

gDrawArc% *Function*

Action Draws an arc or curve to the display.

Syntax **gDrawArc%***(x1!,y1!,radius!,colr&,arcstart!,arcend!)*

Remarks The following parameters are passed to the function:

x1!,y1! Single Integer values indicating the center of the arc in pixels.

radius! Single Integer value indicating the radius of the arc in pixels.

colr& Long Integer value indicating the foreground color of the arc.

arcstart! Single Integer value indicating the starting position of the arc in radians. If arcstart! is negative the angle is treated as positive and draws a radius to start.

arcend! Single Integer value indicating the ending position of the arc in radians. If arcend! is negative the angle is treated as positive and draws a radius to end.

NOTE: To convert values from degrees to radians, multiply the angle (in degrees) by pie/180 (which equals .0174553282519943З).

(see next page for example)

Example

```
xl! = 100
yl! = 100
radius! = 25
colr& = bwEV(0).Gray
arcstart! = 3.14
arcend! = (270 * .01745532925199433)

retcode% =
        gDrawArc%(xl!,yl!,radius!,colr&,arcstart!,arcend!)
```

gDrawCircle% *Function*

Action Draws a circle to the display.

Syntax **gDrawCircle%***(x1!,y1!,radius!,colr&,fillflag%)*

Remarks The following parameters are passed to the function:

x1!,y1! Single Integer values indicating the center of the circle in pixels.

radius! Single Integer value indicating the radius of the circle in pixels.

colr& Long Integer value indicating the foreground color of the circle.

fillflag% Integer value flag indicating if the circle should be filled or not. If True the circle will be filled with the foreground color. *(Note: This attribute is not supported in the standard library driver.)*

NOTE: When using a logical operator with this function only XOR has and effect, and part of the circle may not be displayed.

Example

```
x1! = 100
y1! = 100
radius! = 25
colr& = bwEV(0).Gray
fillflag% = 0

retcode% = gDrawCircle%(x1!,y1!,radius!,colr&,,fillflag%)
```

gDrawEllipse% *Function*

Action Draws an ellipse to the display.

Syntax **gDrawEllipse%***(x1!,y1!,rx!,ry!,colr&,fillflag%)*

Remarks The following parameters are passed to the function:

x1!,y1! Single Integer values indicating the center of the ellipse in pixels.

rx! Single Integer value indicating the x radius of the ellipse in pixels.

ry! Single Integer value indicating the y radius of the ellipse in pixels.

colr& Long Integer value indicating the foreground color of the ellipse.

fillflag% Integer value flag indicating if the ellipse should be filled or not. If True the ellipse will be filled with the foreground color. *(Note: This attribute is not supported in the standard library driver.)*

NOTE: When using a logical operator with this function only XOR has and effect, and part of the ellipse may not be displayed.

Example

```
x1! = 100
y1! = 100
rx! = 25
ry! = 15
colr& = bwEV(0).Gray
fillflag% = 0

retcode% = gDrawEllipse%(x1!,y1!,rx!,ry!,colr&,fillflag%)
```

gDrawLine% *Function*

Action Draws a line to the display.

Syntax **gDrawLine%***(x1%,y1%,x2%,y2%,colr&)*

Remarks The following parameters are passed to the function:

x1%,y1% Integer values indicating the starting position of the line in pixels.

x2%,y2% Integer values indicating the ending position of the line in pixels.

colr& Long Integer value indicating the foreground color of the line.

A solid line is the default line drawing style. To change the line drawing style use the environment property **bwEV(0).Linestyle**. The argument is a 16-bit integer mask used to put pixels on the display. Use a hex value to adjust the line style, some values are listed below:

&HCCCC	Dotted	....
&HF0F0	Dashed	----------------------
&HFF00	Wide Dash	- - - - - - - - - - - - -

The drawing line style is reset to solid &HFFFF after each call to the line function.

To use a logical operator with a line drawing set the environment property **beEV(0).Logicoper**.

0=PSET
1=OR
2=AND
3=XOR

Note that the logical operator is reset to PSET after each call and that Clipping and WCS are only in use with PSET.

Example

```
x1% = 100
y1% = 100
x2% = 300
y2% = 300
colr& = bwEV(0).White

retcode% = gDrawLine%(x1%,y1%,x2%,y2%,colr&)
```

gDrawPCXFile% *Function*

Action Loads and draws a 16 color PCX file to the display.

Syntax **gDrawPCXFile%***(filename$,x1%,y1%,x2%,y2%,border%,_ bcolr&)*

Remarks This procedure will load a 16 color raster image file which has been saved in a PCX file format into a rectangular area on the display. If the image rectangle boundary is smaller than the image file, the image will be clipped. **Note:** The PCX image file must be a 16 color format for proper operation. The following parameters are passed to the function:

filename$ String holding the PCX file name, the drive and directory may be included.

x1%,y1% Integer values indicating the upper left corner of the image rectangle in pixels.

x2%,y2% Integer values indicating the lower right corner of the image rectangle in pixels.

border% Integer value indicating if a border should be displayed around the image. Set to True (-1) if you wish to have a border.

bcolr& Long Integer value indicating the border color

Example

```
filename$ = 'EGUIDEMO.PCX'
x1% = 100
y1% = 100
x2% = 300
y2% = 300
border% = -1          'Draw a Border
bcolr& = bwEV(0).Blue

      retcode%=gDrawPCXFile%(filename$,x1%,y1%,x2%,y2%,bor
      der%,bcolr&)
```

gDrawRect% *Function*

Action Draws a rectangle to the display.

Syntax **gDrawRect%***(x1%,y1%,x2%,y2%,colr&,fillflag%)*

Remarks The following parameters are passed to the function:

x1%,y1% Integer values indicating the starting position of the line in pixels.

x2%,y2% Integer values indicating the ending position of the line in pixels.

colr& Long Integer value indicating the foreground color of the line.

fillflag% Integer value indicating if the rectangle should be filled or not. If True the rectangle is filled with the foreground color.

A solid line is the default line drawing style. To change the line drawing style use the environment property **bwEV(0).Linestyle**. The argument is a 16-bit integer mask used to put pixels on the display. Use a hex value to adjust the line style, some values are listed below:

&HCCCC	Dotted	....
&HF0F0	Dashed	----------------------
&HFF00	Wide Dash	- - - - - - - - - - - - -

The drawing line style is reset to solid &HFFFF after each call to the line function.

To use a logical operator with a line drawing set the environment property **beEV(0).Logicoper**.

0=PSET
1=OR
2=AND
3=XOR

Note that the logical operator is reset to PSET after each call and that Clipping and WCS are only in use with PSET.

If the **fillflag%** parameter is **True** then the Linestyle and Logicoper properties are ignored.

Example

```
x1% = 100
y1% = 100
x2% = 300
y2% = 300
colr& = bwEV(0).White
fillflag% = 0

retcode% = gDrawRect%(x1%,y1%,x2%,y2%,colr&,fillflag%)
```

gDrawText3D% *Function*

Action Draws text to the display window using the pixel coordinate method in a three dimensional format.

Syntax **gDrawText3D%***(col%,row%,text$,acckeypos%,colr&)*

Remarks This function uses the pixel coordinate method for locating text on the screen. The text will be drawn using the active environment font number. The following parameters are passed to the function:

col% Integer value indicating the column to display the text at. This may be any number within the horizontal screen pixel resolution (0-639).

row% Integer value indicating the line to display the text at. This may be any number within the vertical screen pixel resolution (i.e. for screen mode 12 a number from 0-479, for mode 9 a number from 0-349).

text$ String value indicating the text to draw (max=80 characters).

acckeypos% Integer value indicating if the accelerator key position, or set to 0 for none.

Example: File Name

colr& Long Integer value indicating the foreground and shadow color to paint the text, or -1 for the default color.

NOTE: *Use the multi color formula to set the color value. Also if colr&=-1,* **bwEV(0).Gray** and **bwEV(0).Whit e** *are used as the text color. To make the text transparent set* **bwEV(0).FontTrans = -1**.

See Also gSetEnvFontNum%

Example

```
col% = 100
row% = 100
text$ = "Draw Text by 3D Method"
acckeypos% = 14
colr& = -1

retcode%=gDrawTextPix%(col%,row%,text$,acckeypos%,_
                              colr&)
```

gDrawTextCol% *Function*

Action Draws text to the display window using the mixed coordinate method.

Syntax **gDrawTextCol%***(col%,row%,text$,acckeypos%,colr&)*

Remarks This function uses a mixed coordinate method for locating text on the screen. The text will be drawn using the active environment font number. The following parameters are passed to the function:

col% Integer value indicating the column to display the text at. This must be a number from 1-80. If col% is outside the range, the function will return an error code of 50 (Over Flow) and no text will be displayed.

row% Integer value indicating the line to display the text at. This may be any number within the vertical screen pixel resolution (i.e. for screen mode 12 a number from 0-479, for mode 9 a number from 0-349).

text$ String value indicating the text to draw (max=80 characters).

acckeypos% Integer value indicating if the accelerator key position, or set to 0 for none.

Example: File Name

colr& Long Integer value indicating the foreground and background color to paint the text, or -1 for the default color.

NOTE: *Use the multi color formula to set the color value. Also if colr&=-1,* **bwEV(0).FontForeColr** and **bwEV(0).FontBackColr** *are used as the text color. To make the text transparent set* **bwEV(0).FontTrans = -1**.

See Also

gSetEnvFontNum%

Example

```
col% = 10
row% = 100
text$ = "Draw Text by Column Method"
acckeypos% = 14
colr& = (bwEV(0).Black + (bwEV(0).Gray * 256)

retcode%=gDrawTextCol%(col%,row%,text$,acckeypos%,_
                                 colr&)
```

gDrawTextPix% *Function*

Action Draws text to the display window using the pixel coordinate method.

Syntax **gDrawTextPix%***(col%,row%,text$,acckeypos%,colr&)*

Remarks This function uses the pixel coordinate method for locating text on the screen. The text will be drawn using the active environment font number. The following parameters are passed to the function:

col%	Integer value indicating the column to display the text at. This may be any number within the horizontal screen pixel resolution (0-639).
row%	Integer value indicating the line to display the text at. This may be any number within the vertical screen pixel resolution (i.e. for screen mode 12 a number from 0-479, for mode 9 a number from 0-349).
text$	String value indicating the text to draw (max=80 characters).
acckeypos%	Integer value indicating if the accelerator key position, or set to 0 for none. *Example:* File Name
colr&	Long Integer value indicating the foreground and background color to paint the text, or -1 for the default color.

NOTE: *Use the multi color formula to set the color value. Also if colr&=-1,* **bwEV(0).FontForeColr** and **bwEV(0).FontBackColr** *are used as the text color. To make the text transparent set* **bwEV(0).FontTrans = -1**.

See Also

gSetEnvFontNum%

Example

```
col% = 100
row% = 100
text$ = "Draw Text by Pixel Method"
acckeypos% = 14
colr& = (bwEV(0).Black + (bwEV(0).Gray * 256)

retcode%=gDrawTextPix%(col%,row%,text$,acckeypos%,_
                                  colr&)
```

gDropDownBox% *Function*

Action Displays a Pop-Up List Box for the user to select and item from.

Syntax **gDropDownBox%***(cx1%,cy1%,SelectedItem$,ItemList$(),_ reserved%)*

Remarks This procedure functions much like a control, except that it is only visible while it has the focus. After calling the procedure a Pop-Up Box will appear at the specified location with a list of items for the user to select from. If the user clicks the mouse outside of the list box on another control or presses the ESC or ENTER key the box will Pop-Down returning to the calling procedure. While the Pop-Up List Box has the focus it functions the same as a standard list box. **Important:** In addition to the parameters passed to this procedure the Edit Box Properties are also used to control and configure a Pop-Up List Box. See the example below for more information.

x1% Integer values indicating the column to display the Pop-Up List Box. This should be a value between (1-79). **Note:** *A Pop-Up List Box may be placed outside of a Dialog Box Border.*

y1% Integer values indicating the upper left corner row of the Pop-Up List Box in pixels.

SelectedItem$ String holding the selected item if the enter key is pressed. Note: If the List Index Pointer is need it will be in the property **bwLB(0).aptr** on return.

ItemList$() Variable Length String Array holding the list of Items to select. This array must be dimensioned prior to calling the procedure. Also the each element of the array should be padded to the same length.

Reserved% This variable is reserved for future use.

Example

```
'This procedure places a Pop-Up List Box 50 pixels inside of
'the upper left corner of the Dialog Box

  cx1% = x1% + 50                   'x1%,y1% is upper left corner
  cy1% = y1% + 50                   'of Dialog Box

  bwEC(0).col = (cx1% / 8)          'Use the Edit Box Properties
  bwEC(0).row = cy1%                'configure the Pop-Up List Box
  bwEC(0).dlen = 12
  bwEC(0).num2disp = 3

  REDIM ItemList$(5)                       'List should be padded
  ItemList$(1) = "First Item  "            'to equal lengths
  ItemList$(2) = "Second Item "
  ItemList$(3) = "Third Item  "
  ItemList$(4) = "Fourth Item "
  ItemList$(5) = "Fifth Item  "

  retcode% = gDropDownBox%(cx1%, cy1%, SelectedItem$,_
                                       ItemList$(),reserved%)
```

gGetCurrentPath% *Function*

Action Get the currently active Drive and Directory Path.

Syntax **gGetCurrentPath%***(CurrentPath$)*

Remarks The following parameters are return from the function:

CurrentPath$	String with the current active drive and directory.

Error If an Error occurs the DOS Error code will be returned.

Example

```
retcode% = gGetCurrentPath%(CurrentPath$)
```

gGetDirDrvList% *Function*

Action Get a list of directories which exist in the currently active directory path, and a list of valid drives.

Syntax **gGetDirDrvList%***(DirList$(),DirSpec$,Sortflag%)*

Remarks The *DirList$* array must be dimensioned before calling the function with the **REDIM** statement. The maximum directories that may be returned is 112. The following parameters are passed to the function:

DirList$ An array to hold the list of directories and drives which will be returned.

DirSpec$ A String indicating the directory search parameters.

Sortflag% Integer value indicating if sorting should be performed on the array. Set to -1 if you want to sort the array ascending and -2 for sorting the array descending.

Example

```
REDIM DirList$(0)
DirSpec$ = "*."
Sortflag% = 0

retcode% =
gGetDirDrvList%(DirList$(),DirSpec$,Sortflag%)
```

gGetDlbClick% *Function*

Action Get a mouse double click action if it occurs in the click area during the double click duration time.

Syntax **gGetDlbClick%***(x1%,y1%,x2%,y2%,buttonnumber%)*

Remarks The following parameters are passed to the function:

x1%,y1%	Integer values indicating the upper left corner of the click area in pixels.
x2%,y2%	Integer values indicating the lower right corner of the click area in pixels.
buttonnumber%	Integer value indicating the mouse button number to watch. 0=Left Button 1=Right Button 2=Middle Button *(If Present)*

Example

```
x1% = 100
y1% = 100
x2% = 300
y2% = 300
buttonnumber% = 0

retcode%=gGetDlbClick%(x1%,y1%,x2%,y2%,buttonnumber%)
```

gGetFileList% *Function*

Action Get a list of files which exist in the currently active directory path.

Syntax **gGetFileList%***(FileList$(),FileSpec$,Sortflag%)*

Remarks The *FileList$* array must be dimensioned before calling the function with the **REDIM** statement. The maximum files that will be returned is 512. The following parameters are passed to the function:

FileList$	An array to hold the list of files which will be returned.
FileSpec$	A String indicating the file search parameters.
Sortflag%	Integer value indicating if sorting should be performed on the array. Set to -1 if you want to sort the array ascending and -2 for sorting the array descending.

Example

```
REDIM FileList$(0)
FileSpec$ = "*.*"
Sortflag% = 0

retcode% =
gGetFileList%(FileList$(),FileSpec$,Sortflag%)
```

gGetImage% *Function*

Action Gets a image from a rectangular area of the display.

Syntax **gGetImage%***(x1%,y1%,x2%,y2%,array%())*

Remarks The image array must be smaller than 32k. The following parameters are passed to the function:

x1%,y1% Integer values indicating the upper left corner of the area to capture in pixels.

x2%,y2% Integer values indicating the lower right corner of the area to capture in pixels.

array%() Integer array to hold the image information. This array must be dimensioned large enough to hold the image. Use the formula below to calculate the proper size.

Image Array Formula:
asize%=4+INT(((x2%-x1%+1)*1+7)/8)*4*((y2%-y1%)+1)

Example

```
x1% = 100
y1% = 100
x2% = 200
y2% = 200

asz%=4+INT(((x2%-x1%+1)*1+7)/8)*4*((y2%-y1%)+1)
REDIM array%(asz%)

retcode% = gGetImage%(x1%,y1%,x2%,y2%,array%())
```

gGetKeyPress% *Function*

Action Get the current key press, if any, that exists in the keyboard buffer.

Syntax **gGetKeyPress%**

Remarks This function returns the actual key code that was pressed, or a -1 if the ALT KEY is pressed. If the key press is an extended key, such as ALT-L then a negative key scan code is returned, for example:

ALT-L = -38
ALT-C = -46

NOTE: If the ALT KEY is being pressed while the function is being called, it will not return control to the calling procedure until the key is released.

Example

```
'Check For Key Press ------------------------
 KeyPress% = gGetKeyPress%
 IF KeyPress% <> 0 THEN        'Process Key Strokes
        'PLACE CODE TO TAKE ACTION ON HERE
 END IF
```

gGetMouse% *Function*

Action Return the current vertical and horizontal pixel location of the mouse and which button is being pressed.

Syntax **gGetMouse%***(mx%,my%,mb%)*

Remarks To process mouse events you must no the location and button status of the mouse prior to taking action. There are no parameters passed to this procedure, only returned.

NOTE: The **gGetMouse%** parameters *mx%,my%,mb%* are global to the **EGUI System**, so information returned from a get should be used immediately following the call or the information may change.

mx%	Integer value indicating the horizontal pixel location of the mouse.
my%	Integer value indicating the vertical pixel location of the mouse.
mb%	Integer value indicating which mouse button, if any, are currently being pressed. 1=Left Button 2=Right Button 3=Middle Button *(If Present)*

See Also **gShowMouse%,gHideMouse%**

Example

```
retcode% = gGetMouse%(mx%,my%,mb%)
```

gGetSysFontHgt% *Function*

Action Gets the current EGUI System font height in pixels.

Syntax **gGetSysFontHgt%***(fontnumber%)*

Remarks Gets the current active system font height which has been set by **gSetEnvFontNum%**. The System font width is always 8 pixels.

fontnumber% Integer value holding the current font height in pixels.

See Also **gSetEnvFontNum%**

Example

```
        retcode% = gGetSysFontHgt%(fontnumber%)
```

gHandMouse% *Function*

Action Set the mouse pointer to a hand cursor.

Syntax gHandMouse%

See Also gStandardMouse%,gHourGlassMouse%

Example

```
retcode% = gHandMouse%
```

gHideMouse% *Function*

Action Turn the mouse cursor off.

Syntax **gHideMouse%**

Remarks The mouse cursor **must** be turned off before displaying something to the screen.

NOTE: All **EGUI** display functions and controls are mouse sensitive and **DO NOT** require hiding or showing the mouse cursor before or after a call. However any procedure used, that does a direct write to the display should turn the mouse on & off during the writes.

See Also **gShowMouse%,gInstallMouse%**

Example

```
retcode% = gHideMouse%
```

gHourGlassMouse% *Function*

Action Set the mouse pointer to a hourglass cursor.

Syntax gHourGlassMouse%

See Also gStandardMouse%,gHandMouse%

Example

```
retcode% = gHourGlassMouse%
```

glnitBWSystem% *Function*

Action Initialize the **EGUI Environment System.**

Syntax **glnitBWSystem%**

Remarks The EGUI Environment System <u>**must**</u> be initialized before any calls to the GUI Library are made.

NOTE: This code will be built by the EGUI Form Generator when you click the Build Module button on the Build Form Function Window.

Example

```
retcode% = gInitBWSystem%
```

gInstallMouse% *Function*

Action Check for the existence of the mouse and reset the mouse driver to its' default startup parameters.

Syntax **gInstallMouse%**

Remarks The mouse must be installed prior to making any calls to the mouse driver. A mouse install is done during the EGUI System Install and is usually not required at any other time.

See Also **gShowMouse%,gHideMouse%,gGetMouse%**

Example

```
retcode% = gInstallMouse%
```

gLoadIcon% *Function*

Action Load and display an icon file.

Syntax **gLoadIcon%***(filename$,x1%,y1%,,logicflag%)*

Remarks The following parameters are passed to the function:

filename$ String indicating the file name, and optional path, of the icon to load and display.

x1%,y1% Integer values indicating the upper left corner to position the icon on the display, in pixels.

logicflag% Integer value if the icon should be displayed using a logical operator. Available values are listed below:

0=PSET
1=PRESET *(Reverse Image)*
2=OR
3=AND
4=XOR

NOTE: The logical operator is reset to PSET after each call to the function.

Example

```
filename$ = "EDIT.ICN"
ixl% = 100
iyl% = 100
logicflag% = 0

retcode% = gLoadIcon%(filename$,ix1%,iy1%,logicflag%)
```

gLoadSysCfgFile% *Function*

Action Loads the EGUI System Initialization file (EGUI.INI).

Syntax **gLoadSysCfgFile%***(filename$)*

Remarks The EGUI System must be configured before use. Loading the system initialization file sets many of the System Environment Properties and File Paths. **Note: This** process is done in the **gInitBWSystem%** procedure and is normally not called again. *(See System .INI File Format in Chapter 8 for more information.)*

filename$	String value holding the System Initialization File Name **EGUI.INI**.

Example

```
filename$="EGUI.INI"
retcode% = gLoadSysCfgFile%(filename$)
```

gLoadSysFont% *Function*

Action Loads the EGUI System Font file (BWSYS.FNT).

Syntax **gLoadSysFont%***(SysPathInfo$)*

Remarks The EGUI System Font File must be loaded prior to using any fonts. **Note:** This process is done in the **gInitBWSystem%** procedure and is normally not called again.

SysPathInfo$ String value holding the System Path Information. Example: "C:\EGUI"

Example

```
SysPathInfo$="C:\EGUI"
retcode% = gLoadSysFont%(SysPathInfo$)
```

gMessageDialog% *Function*

Action Displays a Dialog Box Window with a Message and prompts the user for an OK,Retry or Cancel reply.

Syntax **gMessageDialog%***(messagelist$(),messfontlist%(),_
messtitle$,messtype%)*

Remarks Use this procedure to display a short message and prompt the user for a simple reply.

messagelist$ Variable length message string array, a maximum of 42 characters per element. This array must be dimensioned prior to calling the procedure.

messfontlist% Integer array which coincides with each element of the message string array and selects the system font type to use with each line. This array must be dimensioned prior to calling the procedure.

messtitle$ A message box title.

messtype% This flag is used to select the type of message box desired.

Type	Icon Displayed	Button
1	Information	OK
2	Warning	OK
3	Critical	Retry, Cancel
4	Error	OK
5	NO ICON	OK, Cancel
6	NO ICON	Yes, No

Note: A Custom Icon can be displayed by making the type negative. Place the Icon File Name to be displayed in messagelist$(0) with the full path of the file.

Example:

```
messagelist$(0)=C:\ICON\CUSTOM.ICN
messtype%=-6    'Display         Yes,No
Buttons
```

Example

```
messtitle$ = "File Message"
messtype% = 1                 'Information Icon with OK
       Button

REDIM messagelist$(3)
messagelist$(1)  = "This is line one."
messagelist$(2)  = "This is line two."
messagelist$(3)  = "This is line three."

REDIM messfontlist%(3)
messfontlist%(1)  = 2         '8x14 Normal
messfontlist%(2)  = 2         '8x14 Normal
messfontlist%(3)  = 2         '8x14 Normal

retcode%=gMessageDialog%(messagelist$(),messfontlist%(),_
                                   messtitle$,messtype%)
```

gMouseCheck% *Function*

Action Check for mouse to make sure no buttons are currently being pressed.

Syntax gMouseCheck%

Remarks This procedure will not return control to the calling procedure until no mouse button is being pressed.

See Also gShowMouse%,gHideMouse%,gMouseFunc%

Example

```
retcode% = gMouseCheck%
```

gMouseFunc% *Function*

Action Call a mouse service routine.

Syntax **gMouseFunc%***(m1%,m2%,m3%,m4%)*

Remarks This procedure allows you to call any mouse service routine that is available through the mouse driver.

See the **Microsoft Mouse Programmers Guide** *for more information on calling mouse services.*

See Also **gShowMouse%,gHideMouse%,gMouseCheck%**

Example

```
'This service sets the mouse range for VGA mode
 640x480
 m1% = 7                   'Function Number 7
 m2% = 0                   'NOT Used
 m3% = 0                   'Upper Left Corner
 m4% = 0

 retcode% = gMouseFunc%(m1%,m2%,m3%,m4%)

 m1% = 8                   'Function Number 8
 m2% = 0                   'NOT Used
 m3% = 639                 'Lower Right Corner
 m4% = 479

 retcode% = gMouseFunc%(m1%,m2%,m3%,m4%)
```

gPaint% *Function*

Action Paints an area on the display.

Syntax gPaint%*(x1%,y1%,colr&)*

Remarks This procedure will fill an area on the display with the selected color starting at the selected point. Painting is complete when a line is painted without changing the color of any pixels. The following parameters are passed to the function:

x1%,y1% Integer values indicating the starting position in pixels.

colr& Long Integer value indicating the foreground color to paint.

Example

```
x1% = 100
y1% = 100
colr& = bwEV(0).Red
retcode% = gPaint%(x1%,y1%,colr&)
```

gPercDoneBarA% *Function*

Action Draw a percentage done bar to the display.

Syntax

```
gPercDoneBarA%(px1%,py1%,pwid%,CurVal!,TotVal!,_
                          Textcolr&, Barcolr&, Border%)
```

Remarks Use this procedure as a progress indicator when the process being performed will require more than a few seconds. This let the user know what is going on and when the process will be complete.

px*1%,py1%* Integer values indicating the upper left corner of the rectangle in pixels.

pwid% Integer value indicating the width of the rectangle in pixels.

CurVal! Single value indicating the current process value.

TotVal! Single value indicating the total count of the process.

Textcolr& Long Integer value indicating the text color. This is a multi color formula. The first color will be used for the first 50% of the process and the second color for the second 50%. If you want the same color through out the process use the same color twice.

Barcolr& Long Integer value indicating the bar color. This is a multi color formula. The first color will be used for the first 50% of the process and the second color for the second 50%. If you want the same color through out the process use the same color twice.

Border% Integer value indicating if a border should be drawn. If the border is drawn it will use the **bwEV(0).Outlinecolr** property for the color.

Example

```
'SetUp Loop Parameters --------------------------------------
 px1% = 100
 py1% = 100
 pwid% = 256
 CurVal! = 0
 TotVal! = 50
 textcolr& = (bwEV(0).White + (bwEV(0).Black * 256))
 barcolr& = (bwEV(0).LightBlue + (bwEV(0).White * 256))
 border% = -1            'Draw Border

 FOR CurVal! = 1 TO TotVal!

   '*************************************************************
   'Put Your Process Here
   '*************************************************************

   'Display Percentage Done ----------------------------------
    retcode% = gPercDoneBarA%(px1%, py1%, pwid%, CurVal!,
               TotVal!,_                      textcolr&,
               barcolr&, border%)

    FOR q% = 1 TO 12000           'This pause for Demo Only
    NEXT q%                       'Remove for actual use

NEXT CurVal!
```

gPutImage% *Function*

Action	Puts an image that has been gotten by **gGetImage%** to the display.
Syntax	**gPutImage%***(x1%,y1%,x2%,y2%,array%(),logicflag%)*
Remarks	The image must have been gotten by the get procedure prior to calling this procedure. The following parameters are passed to the function:

x1%,y1%	Integer values indicating the upper left corner of the area to in pixels.
x2%,y2%	Integer values indicating the lower right corner of the area to in pixels.
array%()	Integer array holding the image information.
logicflag%	Integer value indicating the logical operation to perform on the image. 0=PSET 1=OR 2=AND 3=XOR

See Also gGetImage%

Example

```
x1% = 100
y1% = 100
x2% = 200
y2% = 200
asz%=4+INT(((x2%-x1%+1)*1+7)/8)*4*((y2%-y1%)+1)
REDIM array%(asz%)
retcode% = gGetImage%(x1%,y1%,x2%,y2%,array%())


logicflag%=0
retcode% = gPutImage%(x1%,y1%,x2%,y2%,array%(),logicflag%)
```

gReStartMouse% *Function*

Action Restart the Mouse Driver and restore mouse settings.

Syntax **gReStartMouse%***(mousebuffer$)*

Remarks Use this procedure to enable the mouse driver and restore the mouse buffer settings. The following parameters are passed to the procedure:

mousebuffer$ A copy of the current mouse buffer settings.

NOTE: The function **gStopMouse%** must be called prior to calling **gReStartMouse%** to get a copy of the mouse buffer for restoring the mouse.

See Also **gStopMouse%**

Example

```
'Stop the Mouse and get a copy of its' settings
 retcode% = gStopMouse%(mousebuffer$)

'PLACE YOUR CODE HERE

'Restore the mouse with the setting saved
 retcode% = gReStartMouse%(mousebuffer$)
```

gRemoveDialogBox% *Function*

Action Use this function to remove the currently active Dialog Box and it's control objects from the object list.

Syntax **gRemoveDialogBox%**

Remarks When closing a Dialog Box you must remove it from the screen and erase all of the dialog box control objects from the object list. This should be done in the local exit section of a window, before exiting the function. The exit section is located at the end of the **DBFormat** Main Body.

See Also **gBuildDialogBox%**

Example

```
retcode% = gRemoveDialogBox%
```

gSetEnvFontNum% *Function*

Action Set the active environment font type.

Syntax **gSetEnvFontNum%***(fontnumber%)*

Remarks Use this procedure to select one of the seven different system font types.

fontnumber% Integer value (0-6) which indicates the system font to select.

0=8x16	Bold	(Large)
1=8x14	Bold	(Med)
2=8x14	Normal	
3=8x14	Italic	
4=8x8	Bold	(Small)
5=8x8	Normal	
6=8x8	Italic	

NOTE: The EGUI font system requires a VGA video display adapter or better. If a EGA adapter is active only font 1 and 4 will be available.

Example

```
fontnumber% = 2                    '8x14 Normal
retcode% = gSetEnvFont%(fontnumber%)
```

gSetMouseRange% *Function*

Action Set the mouse movement range.

Syntax **gSetMouseRange%***(x1%,y1%,x2%,y2%)*

Remarks Use this procedure to select one of the seven different system font types.

x1%,y1% Integer values indicating the top left corner of the mouse range in pixels.

x2%,y2% Integer values indicating the bottom right corner of the mouse range in pixels.

Example

```
x1% = 0
y1% = 0
x2% = 639
y2% = 479

retcode% = gSetMouseRange%(x1%,y1%,x2%,y2%)
```

gSetPDMenu% *Function*

Action

Use this function to setup a *Pull Down Menu Contro*l in the Draw Dialog Box Section of the **DBFormat** before calling the *Pull Down Menu Control.*

Syntax

gSetPDMenu%*(menuitems$(),menustatus%(),menuacckey%()*

Remarks

Pull Down Menu items are placed in and array named *menuitems$* which you must dimension to the correct maximum menu size prior to assigning items. The variable *NumofSelection%* is used to dimension a multidimensional array for the pull down list. This variable should be adjusted to a size large enough to accommodate the largest list, however make sure that when the list is pulled down it does not extend past the Dialog Box borders.

The other variable used to dimension the item array is *NumofMenus%* which sets the total number of menu lists.

Use *menustatus%* array to set an item active (-1) or inactive (0), and *menuacckey%* to set and items accelerator key.

Note: See the sample code under **gPullDownMenu%** *Control* in Chapter 12 for and example on how to use this procedure.

See Also

gPullDownMenu%

gSetVideoMode% *Function*

Action Sets a Graphics Display Mode.

Syntax gSetVideoMode%*(Vmode%,ClrScrflag%)*

Remarks This procedure will set the display to a Graphics Mode. The following parameters are passed to the function:

Vmode% Integer values indicating the Graphics Video Mode to set. There are only two legal modes supported in the EGUI Standard Video Driver Library.

Mode 9	EGA 16 Color 640x350
Mode 12	VGA 16 Color 640x480

ClrScrflag% Integer value indicating if the display should be cleared during the set process. Set to True (-1) to have the display cleared. *Note: This setting is ignored by the EGUI Standard Video Driver Library and the display is always cleared.*

Example

```
Vmode% = 12              'VGA 640x480 16 Color Mode
ClrScrflag% = -1
retcode% = gSetVideoMode%(Vmode%,ClrScrflag%)
```

gShowMouse% *Function*

Action Turn the mouse cursor on.

Syntax gShowMouse%

Remarks The mouse cursor should_ be turned on after displaying something to the screen.

NOTE: All **EGUI** display functions and controls are mouse sensitive and **DO NOT** require hiding or showing the mouse cursor before or after a call. However any procedure used, that does a direct write to the display should turn the mouse on & off during the writes.

See Also gHideMouse%,gInstallMouse%

Example

```
retcode% = gShowMouse%
```

gStandardMouse% *Function*

Action Set the mouse pointer to the standard up arrow cursor.

Syntax gStandardMouse%

See Also gHandMouse%,gHourGlassMouse%

Example

```
retcode% = gStandardMouse%
```

gStopMouse% *Function*

Action

Disable the Mouse Driver and save the current mouse settings.

Syntax

gStopMouse%*(mousebuffer$)*

Remarks

Use this procedure to disable the mouse driver and save the mouse buffer settings. The following parameters are returned from the procedure:

mousebuffer$ A copy of the current mouse buffer settings.

NOTE: The function **gStopMouse%** must be called prior to calling **gReStartMouse%** to get a copy of the mouse buffer for restoring the mouse.

See Also

gReStartMouse%

Example

```
'Stop the Mouse and get a copy of its' settings
 retcode% = gStopMouse%(mousebuffer$)

'PLACE YOUR CODE HERE

'Restore the mouse with the setting saved
 retcode% = gReStartMouse%(mousebuffer$)
```

gTitleBar% *Function*

Action Draws or Updates a Title Bar for a Dialog Box.

Syntax **gTitleBar%***(x1%,y1%,x2%,y2%,oprflag%,title$)*

Remarks The following parameters are passed to the function:

x1%,y1% Integer values indicating the top left corner of the Dialog Box, in pixels.

x2%,y2% Integer values indicating the bottom right corner of the Dialog Box, in pixels.

oprflag% Integer value indicating which operation to perform. There are three possible operations:

0 = Draw Title Bar w/Active Dialog Box Color
-1 = Draw Title Bar w/InActive Dialog Box Color
-2 = Update Title Bar Text

NOTE: Operations 0 and -1 are used by **gBuildDialogBox%** function to build a Dialog Box Title Bar, these operations should usually not be needed. The primary usage of this function is to Update the Title Bar with new text (Operation -2).

title$ A string of text to be displayed in the Title Bar. This text will be centered inside of the Title Bar.

(see next page for example)

Example

```
x1% = 100
y1% = 100
x2% = 400
y2% = 300
oprflag% = -2                'Update Title Bar Text
title$ = "Title Bar Text"

retcode% = gTitleBar%(x1%,y1%,x2%,y2%,oprflag%,title$)
```

EGUI Toolkit

Appendix

Object Oriented Programming Techniques

The Microsoft BASIC language is not an Object Oriented Language, however many of the procedures and the BASIC data types which are available will allow for the development of object oriented code. In fact the primary scope of object oriented programming is to organize your software in a collection of discrete objects that incorporate both data structures and behavior. The advantage of using a true object oriented language is that a great deal, if not all, of the additional coding necessary to write an object oriented program in a non-object oriented language is handle by the compiler in the object oriented language.

There are many gains in programming applications using object oriented techniques. Unfortunately there is also some overhead involved when using a non-object orient language. This sometimes discourages conventional programmers from using object oriented techniques. Something important to remember; Object Oriented Programming Techniques are really no more than extensions to Structured Programming Techniques, which the MS BASIC language supports fully. So if you are comfortable with structured programming you will enjoy object oriented programming.

The reason we make reference to this is because the EGUI System uses a great deal of Object Orient Programming Techniques. Please understand that there are to many OOPs techniques to try and cover in this manual. What we want to do is cover some of the techniques used in the EGUI System so you may understand the system better. It is recommended that if you wish to use OOPs techniques for developing your application you should get a good reference manual on the subject, preferably one that refers to object oriented techniques for non-object oriented languages, their are several.

Important: Using the EGUI System does not require the use of OOPs Techniques, but using them will enhance the development of your software.

The following information is directed at programmer using OOPs Techniques so some of the terminology may be foreign to someone not familiar with this type of programming.

We would like to begin by saying that all the User Defined Structures in the EGUI System Include files are perceived as Classes. The EGUI Environment structure is the Super Class of the Control Classes. The Control Classes are subclasses and have their own unique structure depending on the characteristics of the control. Note that some controls share a class, such as the check box and option button controls share the command button class.

The control properties are attributes of the control classes. Some of the properties are associated properties (i.e. paintobj, addobj, etc.). These properties share the same functionality and characteristics, however their actual property values are isolated by their prospective classes.

Most of the procedures found in the library are consider methods. One key feature missing from the BASIC library is the ability to get a pointer to a function procedure. This makes the selection of methods a little more difficult, but you may use a case statement to some what emulate this process.

If you would be interested in more information about the different OOPs techniques used in the EGUI Library please let us know.

Custom Controls

You may write your own custom control to use with the EGUI System. The process of writing a custom control is a little complicated so we have elected to spend a little more time documenting this process in a sperate document. This documentation should be available in the very near future.

If you have the need to write a custom control before this documentation is finished download the source code module **EGUIICON.BAS** from the BBS and take a look at the procedure **gPaintICNGraph%**, this is a custom control. You may also analyze the source code from the EGUI System Library.

Important: Please remember that we supply the source code for the EGUI System Control Objects, however if you need to modify this code please make a copy of the code and modify the copy.

Utilities

◆ CM.EXE (Compiler Manager)

This utility program is very similar to Microsoft's NMake utility. It will allow you to compile and link your application from the command line and it will only recompile the files which have changed since the last compile. If you are building large application with BASIC this utility can be very handy. You are free to use this utility with any programming you may be doing not just when you are working with the EGUI System Library.

To get a Help Screen for the Compiler Manager type CM /H at the command prompt and the following screen should appear.

```
Compiler Manager   Ver 2.01
CopyRight (c) 1989-1991, Mike Bishop, All Rights Reserved.

Syntax: CM [/H /C /L /N] filename or MAK filename

/H - Display Compiler Manager Help Screen
/C - Compile ONLY! Process
/L - Link ONLY! Process
/N - NO Validation Checking

NOTE: If NO parameters are passed to CM on the command line it will
   use the first MAK file in the active directory to compile and link.
   If no MAK file is found CM will abort. If a filename or MAK filename
   has been passed it will be used.  CM first checks for a CM.CFG file
     in the active directory, if not present the DOS Environment is
searched
   for CMMAIN.CFG file and that configuration is used. See CM's
   Documentation for the use of CM.CFG & CMMAIN.CFG.

   The /N switch will force a recompile with no checking of the
   Date & Time Stamps of the source & object files.
```

The CM configuration file is used to set the compile and link configuration information. **Note:** CM requires a copy of the configuration file to run. See the information below about how to configure the configuration file.

Note: The CMMAIN.CFG and the CM.CFG files use the same format. The main configuration is used as a default if the local configuration is not present.

COMPILER=D:\BC7\BIN\BC

The location of the BASIC Compiler you want to use to do the current compile.

BCOPTS=/E/X/O/AH/Fs/FPi/G2/Lr/Ot/S
The switches you want to use with the compiler.

LINKER=D:\BC7\BIN\LINK
The linker you wish to use for the current link.

LINKOPTS=/EX/NOE
The linker options to use.

LINKLIBS=EGUIBC7F+COMPRESS;
The library names of any libraries you want to be searched for during the link process.

[EXENAME=*filename.exe***]**
The executable file name to assign to the program. This is optional. If not present the .MAK or .BAS filename is used.

[STUBFILE=NOCOM+NOLPT+NOEMS]
The name(s) of any BASIC Stub Files you wish to use with the link process. This is optional.

[OVERLAY=*module.bas***]**
The name of any files which should be overlayed. This is optional. **Note:** This option is only compatible with MS PDS 7.x, it should not be used with any other linkers.

• PCX2ICN.EXE

This utility may be used to convert a 16 color .PCX image file to an EGUI .ICN file format.

Syntax:

PCX2ICN *filename*.**PCX** *filename*.**ICN**

Important: The largest and icon image may be is 32k, so if the icon image is larger the process will abort.