DEFT Extra

User's Guide

Tandy Color Computer Software Series

Version 1 First Printing

DEFT Extra User's Guide

DEFT Extra

Software License

DEFT Systems, Inc. grants to you, the customer, a non-exclusive, paid-up license to use the **DEFT Systems** software on **one** computer, subject to the following provisions:

1. Except as otherwise provided in the Software License, applicable copyright laws shall apply to the Software.
2. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to you, but not title to the Software.
3. You may use the Software on one host computer and access that Software through one or more terminals if the Software permits this function.
4. You shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in the Software License. You are expressly prohibited from disassembling the Software.
5. You are permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made.
6. You may resell or distribute unmodified copies of the Software provided you have purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from you.
7. All copyright notices shall be retained on all copies of the Software.

Term

This License is effective until terminated. You may terminate this License at any time by destroying the Software together with all copies in any form. It will also terminate if you fail to comply with any term or condition of the License.

Warranty

These programs, their instruction manual and reference materials are sold AS IS, without warranty as to their performance, merchantability, or fitness for any particular purpose. The entire risk as to the results and performance of these programs is assumed by you.

However, to the original purchaser only, **DEFT Systems, Inc.** warrants the magnetic diskette on which these programs are recorded to be free from defects in materials and faulty workmanship under normal use for a period of thirty days from the date of purchase. If during this thirty day period the diskette should become defective, it may be returned to **DEFT Systems, Inc.** for a replacement without charge, provided you have previously sent in your limited warranty registration notice to **DEFT Systems, Inc.** or send proof of purchase of these programs.

Your sole and exclusive remedy in the event of a defect is expressly limited to replacement of the diskette as provided above. If failure of a diskette has resulted from accident or abuse **DEFT Systems, Inc.** shall have no responsibility to replace the diskette under the terms of this limited warranty.

Any implied warranties relating to the diskette, including any implied warranties of merchantability and fitness for a particular purpose, are limited to a period of thirty days from the date of purchase. **DEFT Systems, Inc.** shall not be liable for indirect, special, or consequential damages resulting from the use of this product. Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitations might not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Support

DEFT Systems, Inc. (and not Radio Shack) is completely responsible for the Warranty and all maintenance and support of the Software. Any questions concerning the Software should be directed to:

DEFT Systems, Inc.
P.O. Box 359
Damascus, Md. 20872

DEFT Extra

1 Introduction

DEFT EXTRA is a set of graphics, gaming and direct I/O routines for use with the **DEFT Pascal Workbench, DEFT Pascal** or **DEFT Bench.** This **DEFT EXTRA User's Guide** describes how to use these routines from Pascal and also contains a technical reference section which describes the assembler interfaces to the **DEFT Pascal** runtime library as well as the format of the **DEFT** object files.

1.1 Diskette Contents

The following files are contained on the distribution diskette:

1. *EXTRA/EXT* - this is a Pascal INTERFACE file which contains all the necessary declarations for using the library.
2. *EXTRA/LIB* - this is an object library which contains all the actual routines. The members of the library are:
   - XGDRAW - contains GDRAW.
   - XGPAINT - contains GPAINT.
   - XGCIRCLE - contains GCIRCLE.
   - XGBOX - contains GBOX
   - XGLINE - contains GLINE
   - XGMAIN - contains GINIT, GDISPLAY, GDISPLAYTEXT, GCLS, GSET and GPOINT
   - XGSUPPRT - contains support routines and tables for use by the preceding graphics modules.
   - XJOY - contains JOYSTICK and JOYFIRE.
   - XSOUND - contains SOUND.
   - XSNDJOY - contains support routines and tables for use by XJOY and XSOUND.
   - XDIRECT - contains OPENDIRECT, POSITIONFILE, UPDATEFILE and APPENDFILE
   - XDIRECTP - contains support routines for XDIRECT.
3. *PAINT/PAS* - this is an example Pascal program that uses the routines in **DEFT EXTRA** to create a crude painting program.

4. *PAINT/OBJ* - this is the object form of PAINT/PAS.
5. *PAINT/BIN* - this is the final binary form of PAINT/PAS linked with EXTRA/LIB.
6. *DISKREC/PAS* - this is an example Pascal program that uses the routines in **DEFT EXTRA** to create a simple file maintenance program.
7. *DISKREC/OBJ* - this is the object form of DISKREC/PAS.
8. *DISKREC/BIN* - this is the final binary form of DISKREC/PAS linked with EXTRA/LIB.

1.2 Using DEFT EXTRA

To use **DEFT EXTRA** you have to do three things:

1. Copy the files EXTRA/EXT and EXTRA/LIB onto your work diskette.
2. Put a **%C EXTRA/EXT** directive at the beginning of your Pascal source program. This will cause the compiler to read in the INTERFACE file which defines all the constants, types, procedures and functions defined in **DEFT EXTRA**. If you want to get a listing of EXTRA/EXT, put a **%L** directive before the %C directive.
3. When you link your program, specify **EXTRA/LIB** as the *last* object file to be linked. This will cause the linker to include all the necessary modules from this library into your final binary program.

1.3 Example PAINT Program

This program uses **DEFT EXTRA** to provide a simple paint facility. After starting the program you will see a blue background with a menu on the left hand side. There will also be single blinking pixel on the screen. You can use the right joystick to move this single pixel cursor about.

To draw something you must select a draw operation and a color to draw with. You select an operation by positioning the cursor within the appropriate box and hitting the fire button. You will hear a beep to indicate that the operation was selected. You select a color by positioning the cursor over the appropriate color bar at the lower

left and hitting the fire button. You can select colors and operations independently. The operations are:

1. Paint in an area with the currently selected color using a border of the currently selected color. You can change the paint color by selecting a new color *after* selecting the paint operation. The border color will remain the same. You can specify the border color by selecting it before selecting the paint operation. To paint, position the cursor to where you want to start painting and hit the fire button.

2. Draw an empty box. Position the cursor at one corner and hit the fire button. Position it to the opposite corner and hit the fire button a second time. The box is then drawn.

3. Draw a full box. Same as draw empty box.

4. Draw a circle. Position the cursor to the center of the circle and hit the fire button. Position it to any point on the circumference and hit it again. The circle is then drawn. The delay that you notice is due to the PAINT program computing the radius of the circle.

5. Draw a line. Position the cursor to a point on the screen and hit the fire button. Position it again and hit the fire button again. The line is drawn and the beginning point of a second line is selected as the ending point of this last line.

6. Drag a line. Position the cursor to a point on the screen and hit the fire button. A single point is set. Move the cursor and a line will automatically be drawn as you move the cursor. Hit the fire button again to stop the operation.

To exit from the program, position the cursor to the extreme upper left-hand corner of the screen and hit the fire button.

2 Graphics

The routines in this section provide the same kinds of graphics capabilities that are found in Extended Basic. These routines allow you to easily draw lines, squares, boxes, circles, ellipses, arcs and odd shaped objects which you can then paint in. These operations as well as point set and interrogate are available in all 8 graphics modes.

2.1 Types and Constants

Before describing each routine, it is necessary to define a number of types and constants that are used by each. The INTERFACE file EXTRA/EXT contains all the definitions of these types and constants.

GMode - This is a type which represents one of the eight graphics modes:

```
TYPE GMode = (G1C, G1R, G2C, G2R, G3C, G3R, G6C, G6R);
```

A complete description of each of these modes can be found in *Getting Started with Color Basic*. No matter which mode that you use, you will be working in a coordinate system that contains 256 horizontal (X) points and 192 vertical (Y) points.

GColor - This is a type which represents one of the eight graphics colors:

```
TYPE    GColor =  (GGreen, GYellow, GBlue, GRed,
                   GBuff, GCyan, GMagenta, GOrange);

CONST   GHiResBlack = GGreen;
        GHiResGreen = GYellow;
        GHiResBuff = GHiResGreen;
```

The first four colors are the primary colors available in the graphics modes that end in **C**. The second four are the alternate colors for those graphics modes. The **GHiRes** colors are the colors available in the graphics modes that end in **R**. GHiResGreen is the primary color and GHiResBuff is the alternate color.

GData - This is a type which represents the standard parameters of a particular graphics mode:

```
TYPE  GData = RECORD
                RowShift    : 2..3;
                PageCount   : 2..12;
                XDivisor    : 1..4;
                YDivisor    : 1..3;
                YShiftCount : 6..8;
                XResolution : 64..256;
                YResolution : 64..192;
                END;
      GDataPtr = ^ GData;
```

where:

- **RowShift** is the log2 value of the number of bytes in a row.
- **PageCount** is the number of 512 byte pages being used.
- **XDivisor** is the amount that the X value should be divided by. It is equal to 256 DIV **XResolution**.
- **YDivisor** is the amount that the Y value should be divided by. It is equal to 192 DIV **YResolution**.
- **YShiftCount** is the log2 value of the **XResolution**.
- **XResolution** is the actual horizontal resolution.
- **YResolution** is the actual vertical resolution.

This information is used within the graphics routines and will usually not be needed by the programmer.

GBlock - This is a type which represents a graphics control block:

```
TYPE  GBlock = RECORD
                 Mode      : GMode;
                 Address   : Integer;
                 AltColor  : Boolean;
                 Table     : GDataPtr;
                 Draw      : RECORD
                   X, Y : Integer;
                   Color : GColor;
                   Angle : 0..7;
                   Scale : 1..127;
                   END;
                 END;
```

Where:

- **Mode** is the initialized graphics mode.
- **Address** is the memory address of the beginning of the graphics data.
- **AltColor** indicates whether the alternate color set is to be used. (TRUE => use alternate)
- **Table** is a pointer to the set of **GData** for this **Mode**.
- **Draw** is a record containing the current DRAW information including current X/Y coordinate, current color, angle adjustment and scaling factor. See **GDraw** for more information.

You will have to declare a variable of type **GBlock** for each graphics screen that you will be using.

2.2 GInit

This routine initializes a variable of type **GBlock**. This is the first routine that you should call. The declaration for **GInit** is:

```
PROCEDURE GInit (VAR GraphBlock : GBlock;
                 GraphMode : GMode;
                 GraphPage : Integer;
                 AltColor : Boolean;
                 Background : GColor;
                 DrawColor : GColor);
```

where:

- **GraphBlock** is your variable of type GBlock which is to be initialized.
- **GraphMode** is the graphics mode that you want to use for this particular graphics area.
- **GraphPage** is a number from 0 to 127 which specifies which 512 byte memory page the graphics area is to start on. The actual memory address used will be equal to this number multiplied by 512. It is important that you select a page which does not overwrite your program or the stack. There are basicly three places that the graphics area can go:

1. Below your program. In this case, you specify an origin to the linker which places your program above the area to be used by the graphics area.

2. Between your program and the stack. In this case, you look at the map produced by the linker to determine the uppermost byte of your program and the stack requirements of your program and then select an area between them that won't overwrite the top of your program, the heap or the bottom of the stack.

   NOTE: If you are going to use **GPaint**, you should not position your graphics page here since **GPaint** uses heap memory for its operation.

3. Above the stack. In this case, you must modify PASBOOT/ASM to start the stack lower in memory and then you can use the memory above this for graphics.

- **AltColor** indicates whether you want to use the primary or alternate color set. TRUE indicates to use the alternate color set.
- **BackGround** is the initial color that you want the graphics area to be set to. **GInit** automatically does a **GCls**.
- **DrawColor** is the initial color that you want to use for **GDraw**.

After calling **GInit** the specified graphics page will be initialized to the **BackGround** color and the values in the **Draw** sub-record of **GraphBlock** will contain the following values:

- **X, Y** will be set to 128,96.
- **Color** will be set to **DrawColor**.
- **Angle** will be set to 0.
- **Scale** will be set to 4.

For example:

```
%C EXTRA/EXT
PROGRAM TextExtra (Input, Output);
VAR Block : GBlock;
  ...
BEGIN
  GInit (Block, G6C, 10, False, GBlue, GYellow);
  ...
```

In this example, **Block** is the local variable that contains the control information for the graphics routines. We are going to use the **G6C** graphics mode which gives us 4 colors and uses 6K of memory. The graphics memory starts at memory location 5120 (10 * 512). We are going to use the primary color set, set the background initially to blue and start drawing with the color yellow.

2.3 GDisplay

This routine causes the specified graphics area to be displayed on the screen. You can have more than one graphics area that your program is working with. However, only 1 at time can be displayed. The declaration is:

```
PROCEDURE GDisplay (VAR GraphBlock : GBlock);
```

For example:

```
GDisplay (Block);
```

2.4 GDisplayText

This routine causes the standard text screen to be displayed. If you do any WRITEs or WRITELNs to the screen after doing a **GDisplay**, your text will be stored in the text screen area, however, it will not be displayed until you call **GDisplayText**. The declaration is:

```
PROCEDURE GDisplayText;
```

For example:

```
GDisplay (Block);
 ...
GDisplayText;
WRITELN ('ENTER X,Y: ');
READLN (X, Y);
GDisplay (Block);
 ...
```

In this example, the first **GDisplay** displays the graphics data. Later in the program when you want to prompt for some information, you use **GDisplayText** to setup the text screen so that you can. After getting the information, **GDisplay** sets the screen back to the graphics mode.

2.5 GCls

This routine causes the graphics area to be cleared to the specified **BackGround** color. The declaration is:

```
PROCEDURE GCls (VAR GraphBlock : GBlock;
                BackGround : GColor);
```

For example:

```
GCls (Block, GRed);
```

2.6 GSet

This routine causes a single point to be set. **X** and **Y** specify the coordinates and **PointColor** specifies what color it should be set to. The declaration is:

```
PROCEDURE GSet (VAR GraphBlock : GBlock;
                X, Y : Integer;
                PointColor : GColor);
```

For example:

```
GSet (Block, 187, 43, GYellow);
```

NOTE: if **X,Y** is outside the range (0..255,0..191) then no pixel is set.

2.7 GPoint

This routine returns the color of the specified point. X and Y specify the coordinates to interrogate. The declaration is:

```
FUNCTION GPoint  (VAR GraphBlock : GBlock;
                  X, Y : Integer) : GColor;
```

For example:

```
IF GPoint (Block, 187, 43) = GYellow THEN ...
```

NOTE: if X,Y is outside the range (0..255,0..191) then the returned value is unpredictable.

2.8 GLine

This routine causes a straight line to be drawn. X1, Y1 are the coordinates of one endpoint and X2, Y2 are the coordinates of the other endpoint. **LineColor** is the color to be used for the line.

```
PROCEDURE GLine (VAR GraphBlock : GBlock;
                 X1, Y1, X2, Y2 : Integer;
                 LineColor : GColor);
```

For example:

```
GLine (Block, X, Y, X+15, Y-15, GGreen);
```

NOTE: if either endpoint is outside of the range (0..255,0..191) the line will not be drawn.

2.9 GBox

This routine causes a box to be drawn. X1, Y1 are the coordinates of one corner of the box and X2, Y2 are the coordinates of the opposite corner. **BoxColor** is the color to use. **Solid** indicates whether the box should be filled in. TRUE indicates fill it in, FALSE indicates don't fill it in. The declaration is:

```
PROCEDURE GBox(VAR GraphBlock : GBlock;
               X1, Y1, X2, Y2 : Integer;
               BoxColor : GColor;
               Solid : Boolean);
```

For example:

```
GBox (Block, X, Y, X+15, Y-15, GGreen, True);
```

NOTE: if either **X1,Y1** or **X2,Y2** are outside the range (0..255,0..191) then the corresponding lines of the box are not drawn.

2.10 GCircle

This routine causes an arc to be drawn. The declaration is:

```
PROCEDURE GCircle (VAR GraphBlock : GBlock;
                   X, Y : Integer;
                   Radius : Integer;
                   Color : GColor;
                   Ratio : Integer;
                   Start, Finish : Integer);
```

where:

- **X, Y** are the coordinates of the center of an ellipse of which the arc is a part.
- **Radius** is the horizontal radius of the ellipse.
- **Color** is the color to use when drawing the arc.
- **Ratio** is the height to width ratio relative to 256. The vertical radius of the ellipse is equal to RADIUS * RATIO DIV 256. Using a ratio of 256 results in an equal radius all the way around. However, due to the pixel layout on the screen, this results in a slight vertical exaggeration. A Ratio of 224 more closely approximates a circle.
- **Start** and **Finish** indicate what portions of the arc that you want drawn. They are numbers in the range 0 to 63 where zero represents the 3 o'clock position, 16 the 6 o'clock position, 32 the 9 o'clock position and 48 the 12 o'clock position.

  You can specify **Start** greater than **Finish** to draw an arc through the 3 o'clock position. A full ellipse is indicated by specifying **Start** and **Finish** to be the same number.

For example:

```
GCircle (Block, 128, 96, 64, GRed, 224, 0, 0);
GCircle (Block, 255, 96, 230, GBlue, 96, 16, 48);
```

The first line draws a red circle in the middle of the screen. The second line draws the left half of an ellipse that is flattened on the top and bottom. This half of an ellipse takes up most of the screen.

NOTE: those portions of the specified arc that lie outside the range of (0..255,0..191) are not drawn.

2.11 GPaint

This routine causes an area of the screen bordered by one color to be filled in with (possibly) another color. **X, Y** are the coordinates of the point at which painting is to begin. **PaintColor** is the color to use while painting. **BorderColor** is the color at which to stop painting.

GPaint uses the heap to allocate small control blocks which indicate where all the outstanding places are to a painted. If there is insufficient memory to complete the operation, then GPaint will return a FALSE. Each control block is 7 bytes long and all are released from the heap when GPaint returns.

```
FUNCTION GPaint  (VAR GraphBlock : GBlock;
                  X, Y : Integer;
                  PaintColor : GColor;
                  BorderColor : GColor) : Boolean;
```

For example:

```
IF NOT GPaint (Block, 15, 80, GRed, GYellow)
THEN ... (* error *)
```

2.12 GDraw

This routine draws lines based on directions contained in an ASCII string. **Directions** is the ASCII string. Since **GDraw** does not modify **Directions**, you can use a constant when calling GDraw. The declaration is:

```
PROCEDURE GDraw(VAR GraphBlock : GBlock;
                VAR Directions : String);
```

The directions string can contain the following characters:

1. A leading **B** which indicates that the immediately following operation is not to result in a line being drawn. If this prefix is not present, then the following movement operation will result in a line being drawn from the current draw position to the new draw position.

2. A leading **N** which indicates that the immediately following operation is not to result in a change in the draw position. This allows you to draw a line and return to the original draw

position. If the **N** prefix is not present, then the end of the specified line becomes the new draw position.

3. An **M** (for move) which indicates that a new X,Y coordinate is to be specified. The number following the **M** is the new **X** coordinate. If a comma follows, then the next number is the new **Y** coordinate. If either number has a leading plus (+) or minus (-) then the corresponding value is relative to the old value. The initial X,Y coordinate position is 128,96.

4. A **U** (up), **D** (down), **L** (left), **R** (right), **E** (upper right), **F** (lower right), **G** (lower left) or **H** (upper left) which indicates a relative move in the specified direction. If a number follows, then it specifies the length of the move. If it is not present, then 1 is assumed.

5. An **A** which indicates that relative moves are to have their direction biased (clockwise) by some multiple of 45 degrees. The number following indicates the bias. If no number is present, then 0 is assumed. The initial default value is 0.

6. An **S** which indicates that relative moves are to be scaled by a specified factor. The number following is divided by 4 and the result is used to scale the amount of movement. The initial default value is 4.

7. A **C** which specifies a particular color to use in following operations. The number following the C is the ordinal value of the type **GColor** to use. The initial value is specified in the **GInit** call.

```
GDraw (Block, 'BM45,30;F3U5C2M+10A1URDLA');
```

The semi-colon (or any illegal character) can be used to improve readability.

3.1 JoyStick

This routine returns one of the coordinate values of one of the joysticks. The declaration is:

```
FUNCTION JoyStick (Select : Integer) : Integer;
```

The value returned is in the range 0..63. The values for **Select** are:

- **0** - right joystick horizontal coordinate.
- **1** - right joystick vertical coordinate.
- **2** - left joystick horizontal coordinate.
- **3** - left joystick vertical coordinate.

The value of the specified coordinate of the specified joystick is returned. For example:

```
X :- JoyStick (0) * 4;
Y := JoyStick (1) * 3;
```

This allows you to get X values in the range 0..252 and Y values in the range 0..189.

3.2 JoyFire

This routine returns an indication of whether the fire button of the specified joystick is being depressed. The declaration is:

```
FUNCTION JoyFire (Select : Integer) : Boolean;
```

The possible values for **Select** are:

- **0** - fire button on right joystick.
- **1** - fire button on left joystick.

A **True** returned value indicates that the button is being depressed. For example:

```
IF JoyFire (0) THEN ... (* do stuff *)
```

3.3 IRandom

This routine returns a pseudo-random integer equally distributed in the range 0..32767. The declaration is:

```
FUNCTION IRandom (VAR Seed : Integer) : Integer;
```