

SOME NOTES ON PROGRAMMING DELTA-CAD

Because a lot of diallists use DeltaCAD, a few projects in this book used Delta CAD macros whose resulting dial plates were printed, cut out, and placed on clay and used as templates. The reason for using DeltaCAD was first because it is popular, second because the printed dial plates can be cut and used over clay to mark hour or calendar lines. The author mostly uses glass, clay, and copper on his outside dials.

The author's first career before winding up as an airline pilot was in programming computers. Some work was commercial, but most work was in operating system software. The first system he programmed was an IBM 1401 in Autocoder. Then an IBM 360 in BAL (Basic Assembler Language), with a bit of PL/I, and FORTRAN. All his work on the IBM 370 and later machines was in BAL and on some other machines he used C and C++. Operating systems used were BOS, MFT, and MVT on the IBM 360s, and VS1 and MVS on the IBM 370s, and GCS under VM .

One pet peeve the author has is that documentation for languages is drawn up by programmers as language specifications, and when the human interface is covered it is always somewhat academic. This tendency became worse when object oriented programming became the standard because the novice is faced with several struggles. Thus the examples in this section for the BASIC macro language for Delta CAD are intended to be "conversational" as opposed to transaction oriented. In other words less object oriented and more of a natural flow. As background, the author designed FIDO and PATCHES which were early spooling systems on IBM 360 mainframes in the 1970s, and TOTO and later SHADOW which were teleprocessing programs running under DOS and MVS on the IBM 370 and later ranges which were sold worldwide from 1972 until 1997. SHADOW in particular made over \$55m in sales before the author lost track and interest.

Older style conversational programs had a natural flow, assumed a sequence of events, and were less well suited to random events controlling the program flow. The next development was transaction oriented programs which used discrete pieces of code invoked when things happened. Less natural, more adaptable to random events controlling program flow, and could still easily be made to look conversational if so desired. Then came object oriented code, here "objects" were defined which had "methods" associated with them which handled "things" that affected the objects. Objects were defined by "classes" with an inheritance structure, and inter-relationships between objects. Thus one simple change here would trigger many "methods" in many "objects" resulting in lots of activity. Hard to make conversational, but highly generalized and ideally suited for random events coming in (a screen input, an interrupt from an outside source), but clearly not a simple natural program flow.

USING A CAD SYSTEM and USING CAD MACROS

A CAD system is a computer program that draws, usually better than the average human. The 2d systems such as Delta CAD are simple to use and provide professional draftings. The author prefers TurboCAD deluxe which is a full 3d modeling system and which was used for most of the pictorials in this series of books. It also has excellent after the fact dimensioning tools. DeltaCAD leads TurboCAD in having the computer do all of the work, including calculating angles and then drawing them. TurboCAD programming is only available in the Professional edition. Computers do what they are told and thus special techniques are needed to do simple things a human can intuitively do, such as constraining a line to the boundaries of a box.

This has a few pointers to help write macros. Writing macros for DeltaCAD is within reach of any computer literate person. To simplify concepts, code in this chapter is extracted and reduced from the programs on the CD or website and thus may not handle some special cases.

This section extracts some code and explains what it is doing, and shows final resulting dial plates. There are many macros available for Delta CAD which are well worth exploring. The author has his own versions of Delta CAD macros on the web site for those who are interested, and these complement the spreadsheets which are also made available. However, these spreadsheets and macros which are fully functional are aimed at education first and foremost, and as a tool second. They are not polished works, they are not intended to produce final polished products, nor are they intended to compete with the excellent macros available.

Programs usually begin with initial setup, then they define variables to hold information being worked on, and they also ask humans what they want, and finally they produce the results.

```
' *****  
' A horizontal dial macro for Delta CAD but in conversational mode as  
' opposed to the more modern object oriented mode, but with notes  
' page numbers refer to Manual.pdf or Basic.pdf provided with delta cad  
' *****  
  
Sub Main()                                ' main procedure is required  
  
' *****  
' Initial house keeping - clear the screen - set the drafting area unit  
' *****  
  
' select all objects that may exist on the screen - p223 of Manual  
' then erase them all - page 189 of Manual  
  
If (dcSelectAll) Then  
    dcEraseSelObjs  
End If  
  
' set the entire future drafting area to inches or centimeters, etc  
' page 43 of the Manual: 1.0 generates inches, and 2.54 is cm  
  
dcSetDrawingScale 0.80
```

At this point, a few thoughts on hard coded numbers, specific dimensions and programming practice are relevant.

Good program practice is not to code numbers inline in a program, rather, those numbers should be in a data definition area. This makes the program easier to change, however, it also makes the program a little harder to understand.

It is similarly bad practice to define display areas, as this program does, of say 0,0 to 1,1 however, if the end result is scalable there is little harm, however good practice would still use symbolic numbers, with those defined in the data definition area.

Programming practice has been for many years to use structured coding techniques, that is IF-END IF, DO-END DO, FOR NEXT, and the like, and never to use the GOTO statements.

Another more recent architecture of programming systems has been to move towards object oriented methods, where an object not only holds the data, but also the methods for altering or reading it. And further, all objects can have inter-relationships, so if something changes that might affect one object, then that and other objects will find out and act accordingly. For example, in Windows bring up two displays of the same folder, and in one display, delete an entry. Object oriented methods are what cause the other folder display to update itself and reflect the first display's changed status.

While BASIC as supplied with Delta CAD is not fully object oriented, it does use some of its concepts, for example the structures needed to talk with humans. Next some programming structures are needed for human interaction. These are not difficult, neither are they quite as simple as elementary BASIC.

www.illustratingshadows.com
extracts from Illustrating Times Shadow

```
' *****  
' A generic definition is required for a screen input area  
' *****  
'  
' Here a box on the screen for user dialog is structurally defined,  
' it is only a definition of the generic area, it does not create it  
' ..... Dialog aaaaa  
'  
' To create the area, there must be a Dim statement making a label  
' relate to this definition  
' ..... Dim bbbbb as aaaaa  
'  
' To use bbbbb there must be a ..... yyy = Dialog(xxxxx) which causes  
' human interaction. So...  
'  
' create an area on the screen starting at x=20, y=20  
' whose size is 200 left to right and 100 top to bottom  
' whose title is "Location" - where 0,0 is top left  
  
Begin Dialog aaaaa 20, 20, 200,100, "Location"  
' the first text string starts at x=5,y=15 on the screen  
' and the text string itself starts at x=60 for a height of 10  
Text 5, 15, 60,10, "Enter latitude"  
  
' the input area starts at x=65 (further right) y=15 (same height) for a  
' size of x=50, y=10  
TextBox 65, 15, 50, 10, .mylat  
  
' the second text string starts at x=5 but now y=30, i.e. lower down  
' and the text string itself starts at x=60 for a height of 10  
Text 5, 30, 60, 10, "Enter longitude"  
  
' and the input area starts at x=65 (further right) y=30  
' (same height) for a size of x=50, y=10  
TextBox 65, 30, 50, 10, .mylng  
  
' the third text string starts at x=5 y=45  
' and the text string itself starts at x=60 for a height of 10  
Text 5, 45, 60, 10, "Enter ref longitude"  
  
' and the input area starts at x=65 y=30 for a size of x=50, y=10  
TextBox 65, 45, 50, 10, .myref  
  
' and two buttons for what the user means, location first, button size  
' next - and all such boxes must have at least one button by the way  
OKButton 65, 65, 40, 10  
CancelButton 65, 85, 40, 10  
End Dialog
```

At this point, a few comments might be helpful. The Begin Dialog has nothing to do with a dialog. It is an encyclopedia definition of what you might wish to actually create.

It is created with the Dim statement.

```
' *****  
' The generic definition must then be generated with a name  
' *****  
'  
' this defines "bbbbb" as an instance of aaaaa dialog  
Dim bbbbb As aaaaa
```

And used later.....

```
' *****  
' Now define the initial general working variables  
' *****  
'  
' define a lat and a long, and a reference longitude  
Dim lat As single  
Dim lng As single  
Dim ref As single
```

... continued on the next page

... continued from the last page

```
' *****  
' Now get the lat, long, and reference longitude  
' *****  
  
' first set the defaults - here bbbbb.mylat uses the structure  
' from aaaaa  
bbbbb.mylat = "32.75"  
bbbbb.mylng = "108.2"  
bbbbb.myref = "105.0"
```

....., in fact here it is being used!

```
' here the dialog is invoked and the button results returned to ccccc  
' page 20 and 24 etc of Basic discusses the Dialog function  
ccccc = Dialog(bbbbbb)  
' which causes the answer to be returned  
lat = bbbbbb.mylat  
lng = bbbbbb.mylng  
ref = bbbbbb.myref  
' CANCEL button returns 0  
' OK button returns -1  
' you can determine the button with - Print ccccc, lat, lng, ref
```

HORIZONTAL DIAL

The rest of the program is straight forward.

```
' *****  
' ok, what was returned? if "ok button" then do the program itself  
' *****  
  
' ccccc = -1 means the ok button was used and not the cancel button  
If ccccc = -1 Then  
  
' *****  
' this is the main program to draw the horizontal dial itself  
' *****  
  
' calculate hour line angles next, but first define them  
Dim h, hx, hy As Single ' Delta CAD is fussy about data attributes  
  
' the formula is... hourlineangle = atan ( sin(lat) * tan (lha) )  
' almost all systems us radians  
' the formula also needs adjustment for longitude displacement  
  
' line color is 0 is black  
' line type is dcsolid  
' line weight is dcnormal  
  
' set the text color, font, size, etc also  
dcSetTextParms dcBLACK,"Ariel","Bold",0,8, 20,0,0 ' p321 of Manual  
dcreatetext -1.25, -0.3, 0, "Hour and hour line angle H-DIAL"  
dcreatetext -1.25, -0.9, 0, "Lat: "  
dcreatetext -0.8, -0.9, 0, Int(lat)  
dcreatetext 0.0, -0.9, 0, "Long: "  
dcreatetext 0.3, -0.9, 0, Int(lng)  
  
For hr = 6 To 18 Step 1  
' =====  
' for the hour (hr) calculate the hour line angle (h)  
h = deg(Atn(Sin(rad(lat))*Tan(rad((hr*15) +(ref-lng))))))  
  
' show the time in hours  
dcSetTextParms dcBLACK,"Ariel","Bold",0,8, 21,0,0 ' p321 Manual  
dcreatetext (-1.2+((hr-6)/5)), -0.5, 0, Abs(hr)  
' show the angle  
dcSetTextParms dcBLACK,"Ariel","Bold",0,6, 21,0,0 ' p321 Manual  
dcreatetext (-1.2+((hr-6)/5)), -0.7, 0, Int(h)
```

www.illustratingshadows.com
extracts from Illustrating Times Shadow

```
If hr < 12 Then
' -----
' morning hours ~ NOTE code for keeping lines in a boxed area
' -----
dcsetlineparms dcblue,dcsolid,dcthin ' page 228 Manual
If Abs(h) < 45 Then ' lines touch top of box
  dcSetTextParms dcBLACK,"Ariel","Bold",0,8,21,0,0 ' p321
  hx = Tan(rad((h)))
  dccreateline 0, 0, hx, 1
  dccreatetext (hx), 1.1, 0, Abs(hr) ' page 187 of Manual
Else ' lines touch side of box
  dcSetTextParms dcBLACK,"Ariel","Bold",0,8, 20,0,0 ' p321
  hy = Tan(rad((90-h)))
  dccreateline 0, 0, -1, -hy
  dccreatetext -1.1, -hy, 0, Abs(hr) ' page 187 of Manual
End If

ElseIf hr = 12 Then
' -----
' noon hours
' -----
... similar code which is straight forward

Else
' -----
' afternoon hours
' -----
... similar code except the sign of the angle goes -ve if 90

End If
' =====
Next h
```

The program concludes with drawing a couple of boxes.

```
' draw a box around everything
dcreatebox -1, 0, 1, 1 ' page 184 Manual
dcreatebox -1.2, -.2, 1.2, 1.2 ' page 184 Manual
dcviewbox -1.1, -1.1, 1.1, 1.3 ' page 225 Manual
End If

' *****
' *** this ends the entire program
' *****
End Sub
```

And after the end of the program, there are the DEGREES and RADIANS functions.

```
' *****
' Useful routines or functions - Functions must be defined at the end
' after the main program which is sub(xx) ... end sub
' *****

' Convert degrees to radians
Function Rad ( n As single ) As single
' page 83 basic.pdf for functions
  Rad = (n * 2 * 3.14159) / 360
End Function

' Convert radians to degrees
Function Deg ( n As single ) As single
' page 83 basic.pdf for functions
  Deg = (360 * n) / (2 * 3.14159)
End Function
```

The actual code on the web site may differ and have code to correct for certain situations, however, the objective here has been to step through most of the procedural coding. There are some coding violations, however they have been somewhat intentional in order to make the process of a complete macro, when accompanied by DeltaCAD's two books, MANUAL.PDF and BASIC.PDF, more understandable.

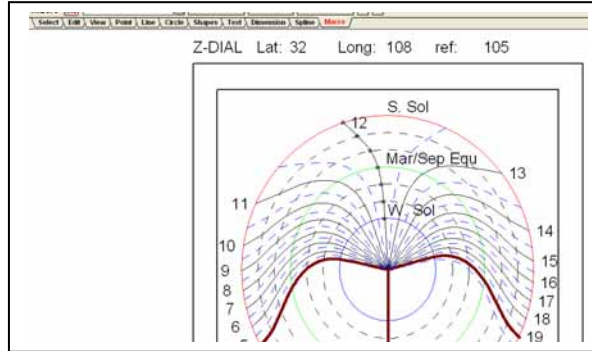
PRINTING TECHNIQUES ~ ~ ~ FOR DIAL PLATES LARGER THAN PRINT PAPER SIZE

Once you have a DeltaCAD macro executed and a dial plate depicted, printing is easy.

FILE

PRINT SETUP

- select printer
- select properties
- landscape or portrait
- OK



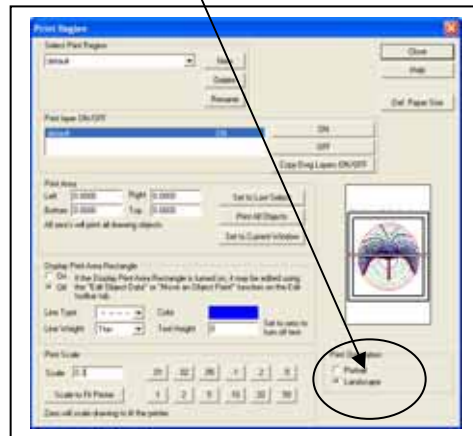
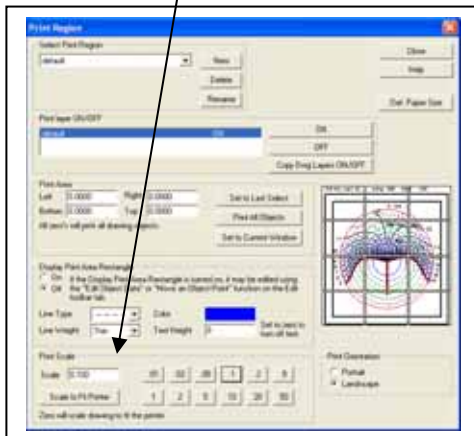
FILE

SET PRINT REGION

- check portrait or landscape
- set print scale so that the result is the final size you want, recall that paper is often 8.5 x 11 inches. The left used a print scale of 0.1 and needed 12 sheets, this would be a big dial, the right used a scale of 0.3 and used 2 sheets and the final dial plate size can be visually estimated.

FILE

PRINT




The above process can be used to print dial plates using DeltaCAD for dials larger than the print paper size. CAUTION: Some screen capture programs do not accurately preserve aspect ratios, so be careful if you use a screen captured layout and paste it into a word processor.

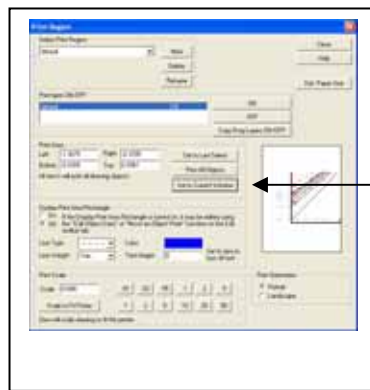
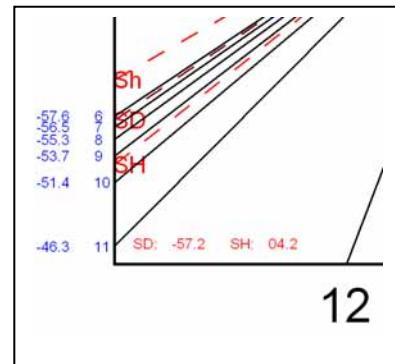
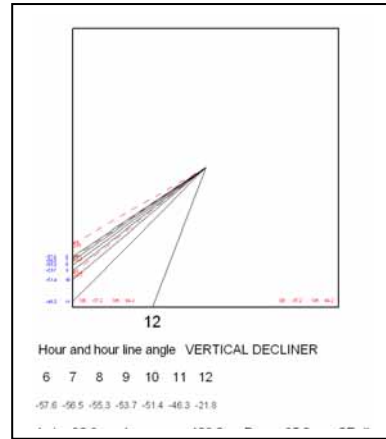
PRINTING TECHNIQUES ~ ~ ~ PRINTING A SMALL PART OF A DIAL PLATE

This is typically declining vertical dials facing about east or west

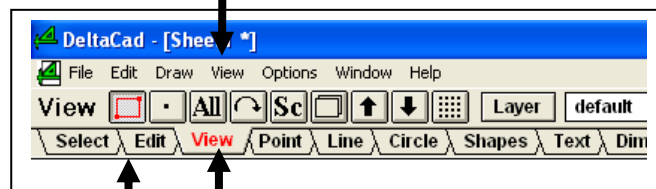
Once you have the dial plate drawn, as shown to the left, use DeltaCAD VIEW (top row, not third row of tabs in DeltaCAD) and then zoom to get the area you desire. If you use screen capture, beware that some capture programs may distort DeltaCAD angles. The angles in DeltaCAD are correct as tabulated and as displayed.

Because of hour line bunching, use the following print technique:

VIEW (third row, not top row) then
VIEW OBJECT IN RECTANGLE and draw its rectangle 
FILE then SET PRINT REGION and in that box...
in PRINTAREA do SET TO CURRENT WINDOW
you may enter print scale numbers also
FILE then PRINT PREVIEW and then you may print



VIEW on top row



VIEW on third row

OBJECT IN RECTANGLE

The above process can be used to print dial plates using DeltaCAD for dials smaller than the displayed size. CAUTION: Some screen capture programs do not accurately preserve aspect ratios, so be careful if you use a screen captured layout and paste it into a word processor.

NOTES ABOUT ISSUES WITH DELTACAD

1. SCRIPT ERROR

NOTE: The command STOP causes a BASIC SCRIPT ERROR, use EXIT FUNCTION or some equivalent instead. **October 25, 2009**

2. TEXT REVERSAL

NOTE: In DeltaCAD prior to version 7, when a CAD drawing was mirrored, then the text location was mirrored, however the text itself stayed un mirrored. This made sense. The mirror function is used in the following macros:

```
calendarDeclination.bas  
MAIN-m-dials.bas  
MAIN-m-dials[f].bas
```

and prior to version 7, they worked well. With version 7, the text itself, not just the location was reversed. While DeltaCAD version 7 allows mirroring not to mirror the actual text itself in a CAD drawing

```
SELECT  
OPTIONS (on right hand side, not on top bar)  
MIRROR TEXT LOCATION ONLY
```

this is cancelled when a macro is started. I changed the macros using mirroring to at least correct some of the text, however, short of using a stack for the other text, sadly the hour line markings are mirrored text. The lines themselves are correct, just the text needs to be rotated. I tried to add a loop that selected each object, and if text then to extract the x,y location and the text, and then delete the object and then add a new object back in with that saved text at the saved x,y location. That didn't work. I emailed the DeltaCAD developer and he said a fix would be coming along either in the next DeltaCAD version or as an update.

3. TYPE MISMATCH

NOTE: In DeltaCAD version 7, the following code works well

```
Dim xxx as single  
xxx = 10  
yyy = mycode(-xxx)
```

but in version 6 it fails with a "TYPE MISMATCH". The solution in the call using the "-xxx" is:-

```
yyy = mycode( (xxx*-1) )
```