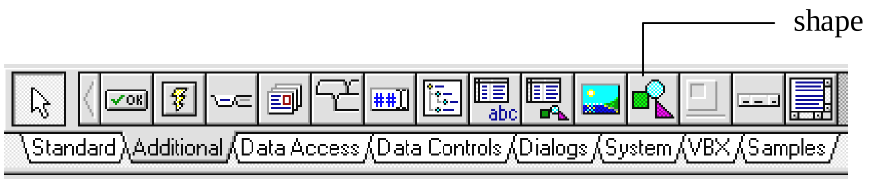
TWO

Graphics

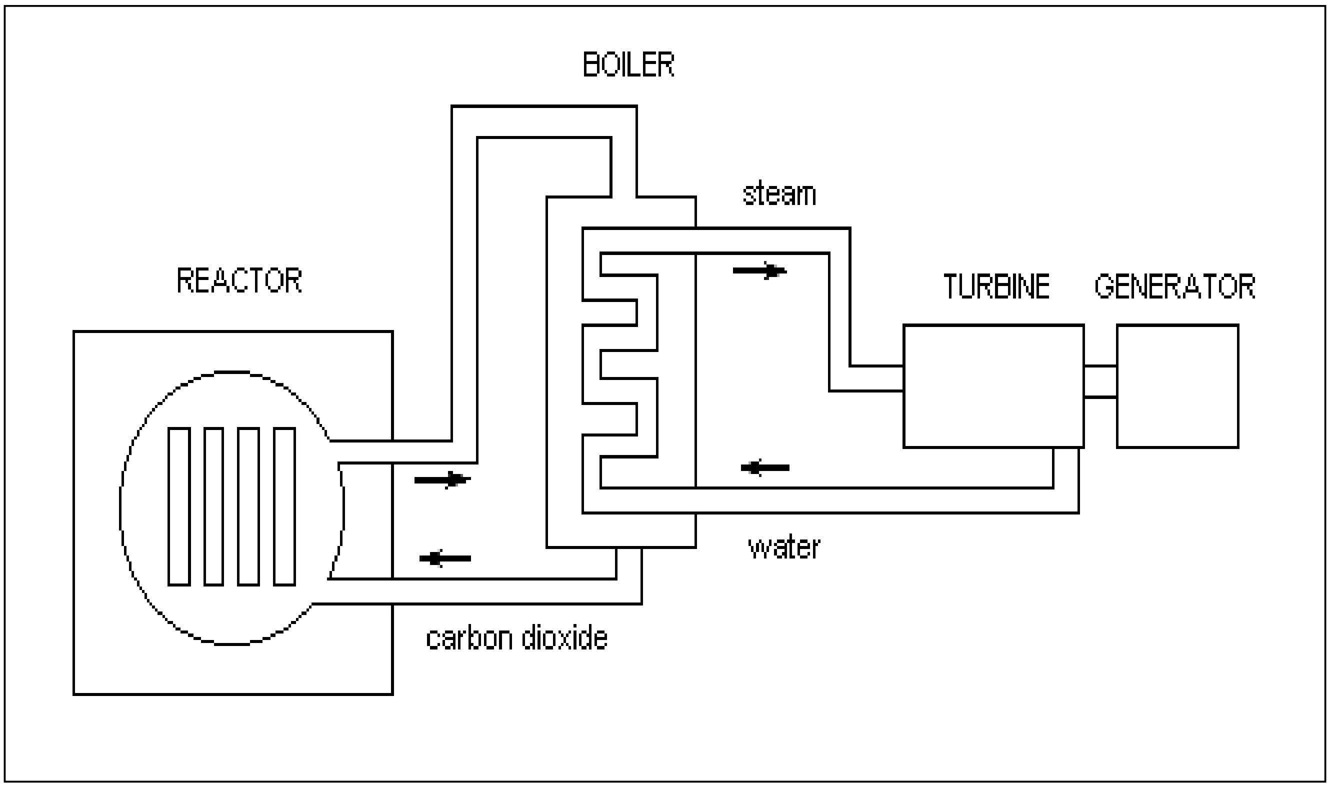
In chapter one we saw how a bitmap image can be displayed in a program. This is not the only way of producing graphics in Delphi, and in this section we will look at two other methods:

Using the *shape* component

Simple diagrams made up of geometrical shapes can be designed directly in Delphi using the *shape* component from the ADDITIONAL component menu:



To try out this technique we will produce a Delphi screen display to illustrate the layout of a nuclear power station:



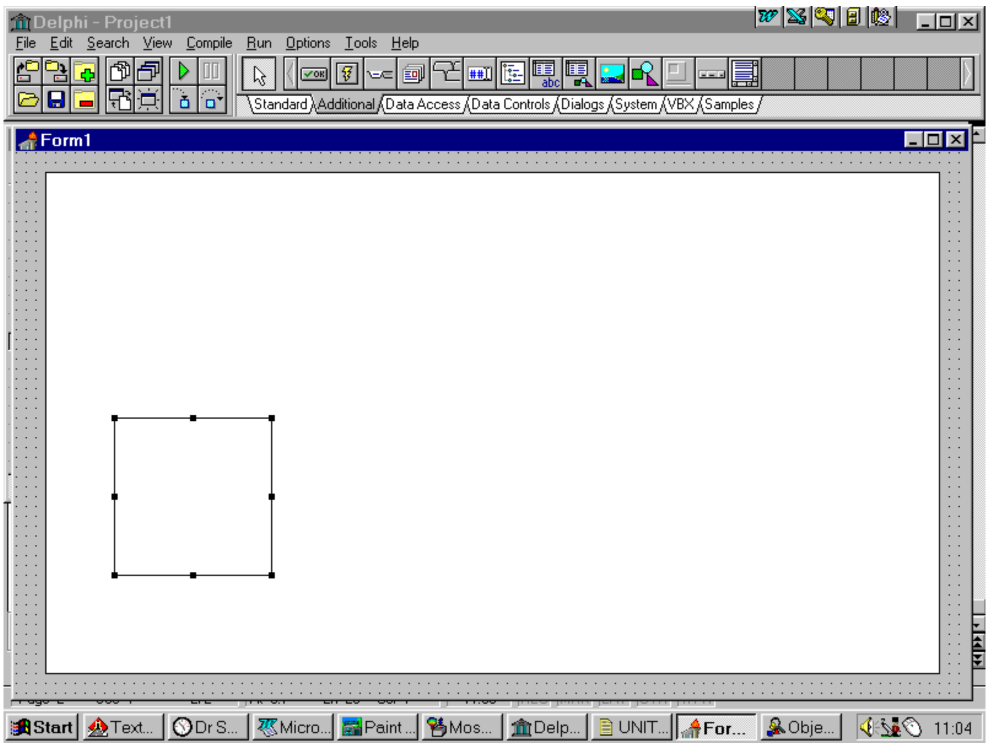
Begin by creating a sub-directory in your work area; call this NUCLEAR.

Load Delphi and immediately save the new project into the NUCLEAR sub-directory. Accept the file names ***unit1.pas*** and ***project1.dpr*** offered by the system. Try to get into the habit of saving your project at regular intervals as you are developing a program - if anything goes wrong, you will have a recent version of the project which can be reloaded.

Using the object inspector, set the *WindowState* to *maximized*. Run the program to check that a blank grey screen in produced. Return to the Delphi editor by clicking the cross in the top right hand corner of the screen.

Use the mouse to drag the edges of the grey dotted grid so that the Form Window nearly fills the screen. Select the *shape* component from the ADDITIONAL menu, and use the mouse to position a large white rectangle so that it nearly fills the Form Window grid - just leave a narrow grey border showing.

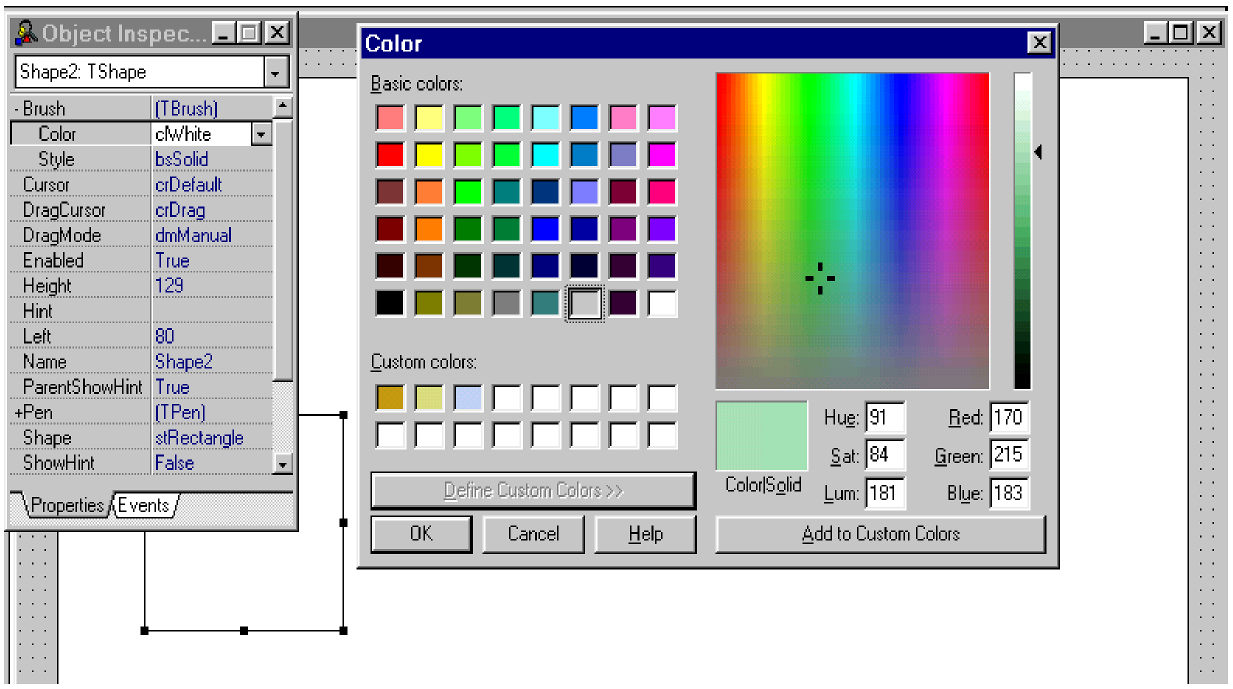
This rectangle is the work area in which we will build up the diagram of the nuclear power station. Begin by selecting the shape component and positioning a square for the reactor outer casing:


Delphi - Project1
File Edit Search View Compile Run Options Tools Help
Standard Additional Data Access Data Controls Dialogs System VBX Samples
Form1
Start Text... Dr S... Micro... Paint... Mos... Delp... UNIT... For... Obje... 11:04

The diagram will look best if it is produced in colour. Make sure that the square is selected by clicking on it with the mouse, then press the ENTER key to display the Object Inspector window.

Double-click on **Brush** to display the sub-properties **Color** and **Style**. Now double-click in the right-hand column alongside the word **Color** and the colour selection window will open. This provides a range of basic colours, and we might select grey as suitable for the concrete shielding of the nuclear reactor. Notice that there is an option to set other colours not in the standard palette; this is done by moving a cross around on the colour chart to specify the colour position in the spectrum, and then moving the pointer on the vertical scale to set the lightness/darkness required.

Once a suitable colour has been chosen, click OK to return to the Form grid. The square should now have the required colour fill.



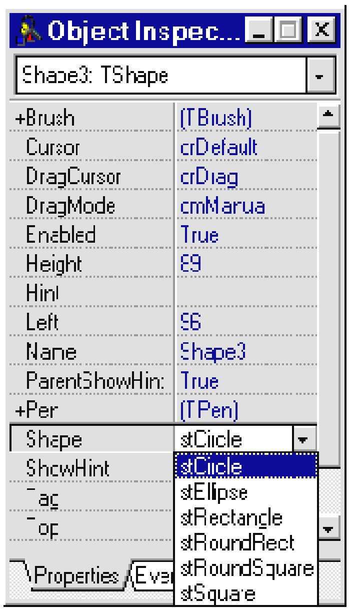
Shape2: TShape

- Brush	(TBrush)
Color	clWhite
Style	bsSolid
Cursor	crDefault
DragCursor	crDrag
DragMode	dmManual
Enabled	True
Height	129
Hint	
Left	80
Name	Shape2
ParentShowHint	True
+Pen	(TPen)
Shape	stRectangle
ShowHint	False

Hue:	91	Red:	170
Sat:	84	Green:	215
Lum:	181	Blue:	183

Next we will add the circle to represent the reactor vessel. Choose the *shape* component and use the mouse to create a square in the position where the circle is to fit.

Bring the Object Inspector onto the screen and find the *Shape* property. Click in the right-hand column to display a small arrow for a drop down menu.



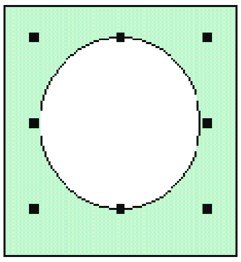
Object Inspec...

Shape3: TShape

+Brush	(TBrush)
Cursor	crDefault
DragCursor	crDrag
DragMode	dmManual
Enabled	True
Height	89
Hint	
Left	96
Name	Shape3
ParentShowHint	True
+Pen	(TPen)
Shape	stCircle
ShowHint	
Tag	
Top	

stCircle
stEllipse
stRectangle
stRoundRect
stRoundSquare
stSquare

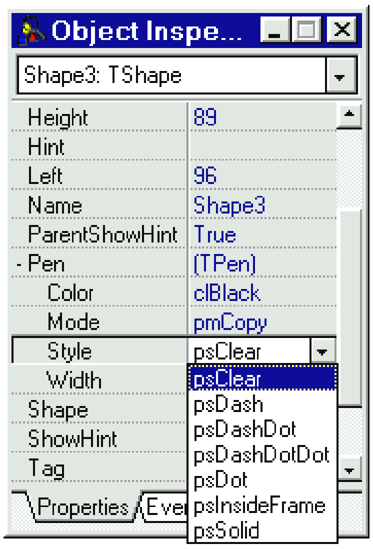
Click the arrow and select *Circle* from the list of shapes offered. The square displayed on the Form grid should now change to a circle shape.



Change the *Brush* property to a suitable colour to represent the carbon dioxide coolant in the reactor vessel.

We can continue to add shapes to the diagram. Put four narrow rectangles inside the reactor vessel to represent the fuel rods, and colour these red. You may find it easiest to use the **Edit/Copy/Paste** facility to do this. Rectangles can be used to represent the turbine and generator.

A problem arises when we try to represent the pipework within the boiler using rectangles. Shapes normally have an outline, in black unless some other colour is selected, but these will spoil the continuity of the pipes.



Object Inspe...

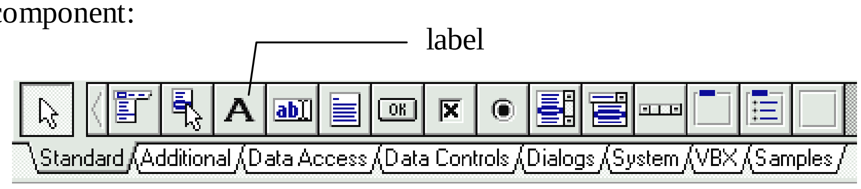
Shape3: TShape

Height	89
Hint	
Left	96
Name	Shape3
ParentShowHint	True
- Pen	(TPen)
Color	clBlack
Mode	pmCopy
Style	psClear
Width	
Shape	
ShowHint	
Tag	

psClear
psDash
psDashDot
psDashDotDot
psDot
psInsideFrame
psSolid

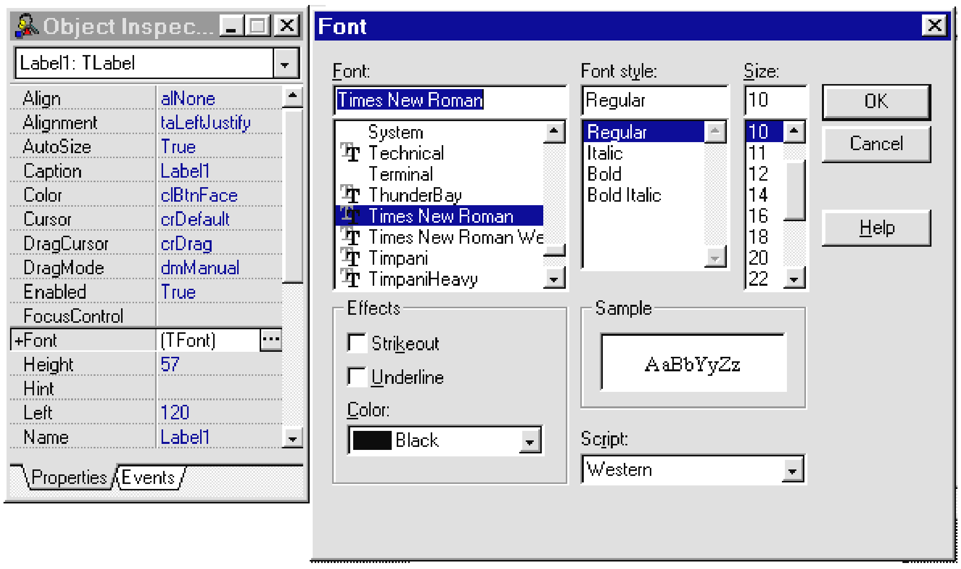
From the Object Inspector, find the **Pen** property. Double-click to produce a list of sub-properties. Click alongside **Style** to obtain a drop down menu then select *Clear*. The shape will be displayed as a coloured bar without any outline. Again, you may find the **Edit/Copy/Paste** a useful way to duplicate shapes, then stretch and drag them into position.

Once the digram shapes are completed, captions can be added using the ***Label*** component:



Select Label, then use the mouse to position the label box on the diagram. Press ENTER to bring up the Object Inspector window and type the required text into the **Caption** property. You will need to set the **Color** property to **White** to produce a white background for the text.

The font, size and colour for the text can be selected by double-clicking in the right-hand column of the **Font** property. A selection window will be displayed:



Object Inspec...

Label1: TLabel

Align	alNone
Alignment	taLeftJustify
AutoSize	True
Caption	Label1
Color	clBtnFace
Cursor	crDefault
DragCursor	crDrag
DragMode	dmManual
Enabled	True
FocusControl	
+Font	(TFont)
Height	57
Hint	
Left	120
Name	Label1

Properties / Events

Font

Font: Times New Roman
System, Technical, Terminal, ThunderBay, Times New Roman, Times New Roman We, Timpani, TimpaniHeavy

Font style: Regular
Regular, Italic, Bold, Bold Italic

Size: 10
10, 11, 12, 14, 16, 18, 20, 22

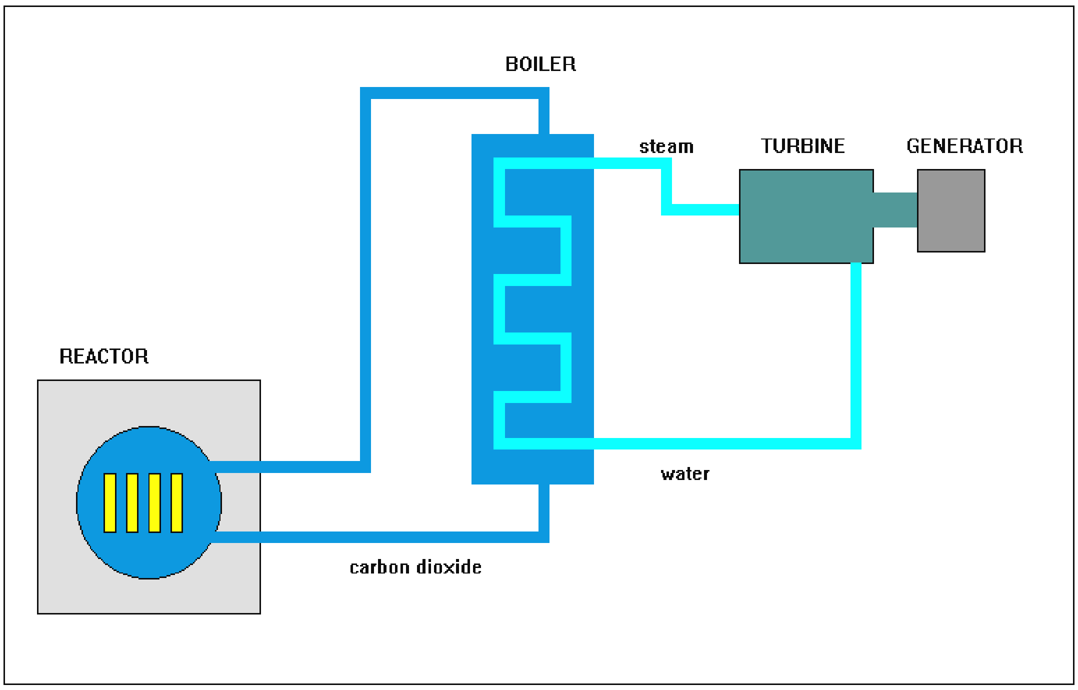
OK / Cancel / Help

Effects
☐ Strikeout
☐ Underline
Color: Black

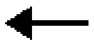
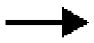
Sample
AaBbYyZz

Script: Western

Complete the labelling for the diagram:



As a final stage you might add arrows to the diagram to show the flow of carbon dioxide and water. Two bitmap images have been provided:


LEFT.BMP

RIGHT.BMP

These can be positioned on the diagram using the **Image** component in a similar way to the camera picture in chapter one, and can be scaled to the required size by setting the **Stretch** property to **true.**

Save the completed project in the sub-directory NUCLEAR.

Run time graphics

The graphical techniques we have used so far have involved setting up screen displays on a Form grid to display bitmap images or shapes. These are known as *design time* graphics because they are created while the program is at the design stage.

Sometimes, however, it is necessary for graphics to be drawn while the program is running - for example in response to the user moving the mouse around the screen in a computer aided design package. In this section we will look at ways of producing *run time* graphics. The technique involves actual programming, rather than just using the component toolbox.

As an example we will write a program to draw a picture of a house. Set up a new sub-directory for the project and name this HOUSE. Start a new Delphi project and immediately save it into the subdirectory.

Go to the Object Inspector for the Form, and set **WindowState** to **Maximized**. Use the mouse to drag the Form grid larger to almost fill the screen.

Add an **Image** box from the ADDITIONAL component menu. Drag the box until it nearly fills the grid.

When we produce the graphics at run time, lines and other shapes will be positioned on the screen using a coordinate system within the image box. We therefore need to set the exact measurements of the image box we are using. Select the image box by clicking the mouse within the frame, then press ENTER to bring up the Object Inspector window. Set the **Height** property to 480:

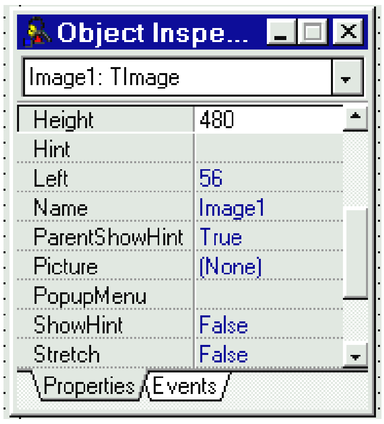
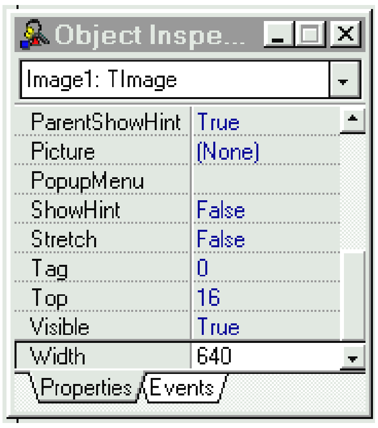


Image1: TImage

Height	480
Hint	
Left	56
Name	Image1
ParentShowHint	True
Picture	(None)
PopupMenu	
ShowHint	False
Stretch	False



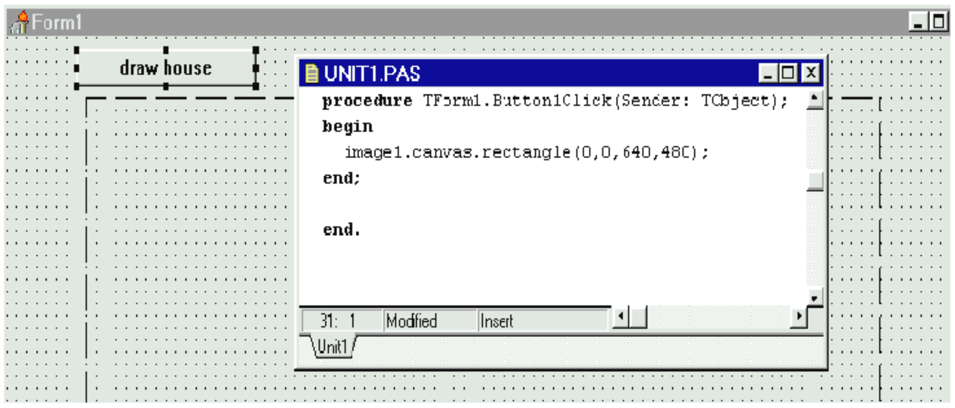
ParentShowHint	True
Picture	(None)
PopupMenu	
ShowHint	False
Stretch	False
Tag	0
Top	16
Visible	True
Width	640

Now set the **Width** property to 640 in a similar way.

You may find that the image box has changed size and you will need to recenter it on the grid.

The house picture we are going to create is drawn out on graph paper on the next page. The size of the grid - 640 units wide and 480 units high - corresponds with the size we set for the image box. Notice that, unlike graphs in mathematics, the vertical scale is numbered <u>downwards</u> from the top of the sheet.

We will begin by putting a button component onto the screen. Pressing this button while the program is running should make the program draw the house. Select the **Button** component and use the mouse to position a button at the edge of the image box. Use the Object Inspector to set the **Caption** property to read 'draw house':



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  image1.canvas.rectangle(0,0,640,480);
end;

end.
```

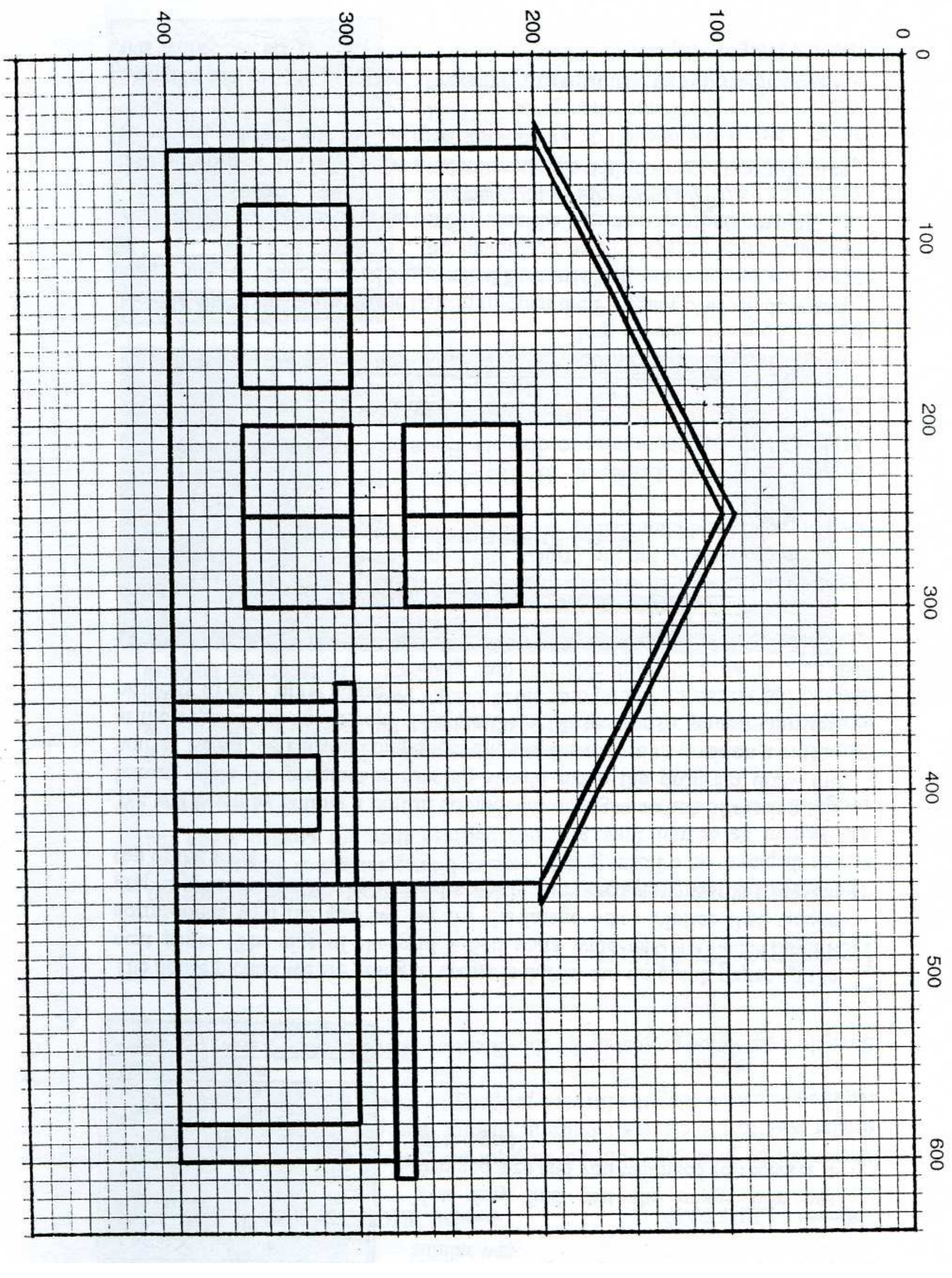
Double-click the mouse on the button to open the Program Unit window and create an event handler procedure called Button1Click.

Add the following line of program between the **begin** and **end** commands:

```
image1.canvas.rectangle(0,0,640,480);
```

The purpose of this is to draw a white rectangle with corners at coordinates (0 across, 0 down) and (640 across, 480 down).

Compile and run the program. Press the '**draw house**' button to check that this works correctly.



0
100
200
300
400
0
100
200
300
400
500
600

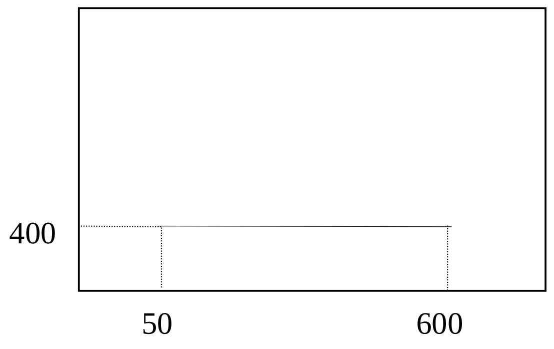
We can now begin to draw the house. Use the '**Toggle form/unit**' short cut button to bring the program unit window to the front, then add two more lines to the **ButtonClick** procedure so that it becomes:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(50,400);
  image1.canvas.lineto(600,400);
end;
```

This should draw the base line for the house. Run the program to check that it works:

- Using the coordinate system, the **moveto** command has told the program to go to a point (50 across, 400 down).
- The **lineto** command has told the program to draw a line from here to a point (600 across, 400 down).

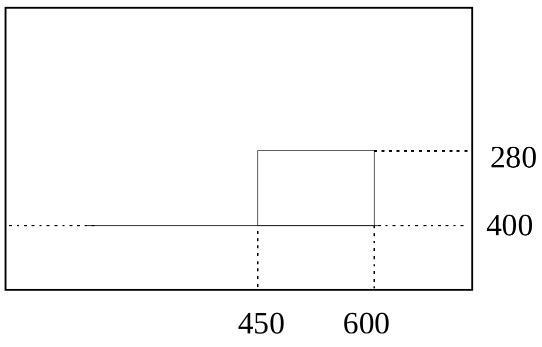
Notice that the *across* coordinate is always given first, followed by the *down* coordinate. Check these numbers on the house drawing on the graph paper.



We might now draw in the garage. This can be done with another rectangle command. Add this line to the procedure:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(50,400);
  image1.canvas.lineto(600,400);
  image1.canvas.rectangle(450,280,601,401);
end;
```

The rectangle which the program will draw has its top left corner at the point (450 across, 280 down). The bottom right corner stops one unit short of the coordinates given, so the actual position of the bottom right corner will be (600 across, 400 down) as required. When using the **rectangle** command, always remember to add one unit to the coordinates for the bottom right corner.



Try to complete the house picture using moveto, lineto and rectangle commands. A few more lines have been added below to help you get started. You may find it easiest to use **Edit/Copy/Paste** to replicate lines of program:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(50,400);
  image1.canvas.lineto(600,400);
  image1.canvas.rectangle(450,280,601,401);
  image1.canvas.rectangle(450,270,611,281);
  image1.canvas.rectangle(470,300,581,401);
  image1.canvas.rectangle(350,310,361,401);
  image1.canvas.moveto(50,400);
  image1.canvas.lineto(50,200);
  image1.canvas.lineto(250,100);
  image1.canvas.lineto(450,200);
  image1.canvas.lineto(450,400);
end;
```

Once the outline drawing is completed, we might think about adding some colour. A convenient way to do this is to use the **floodfill** command. At the bottom of the procedure, just above the **end** command, add the lines:

```
image1.canvas.brush.color:=clRed;
image1.canvas.floodfill(500,350,clBlack,fsBorder);
```

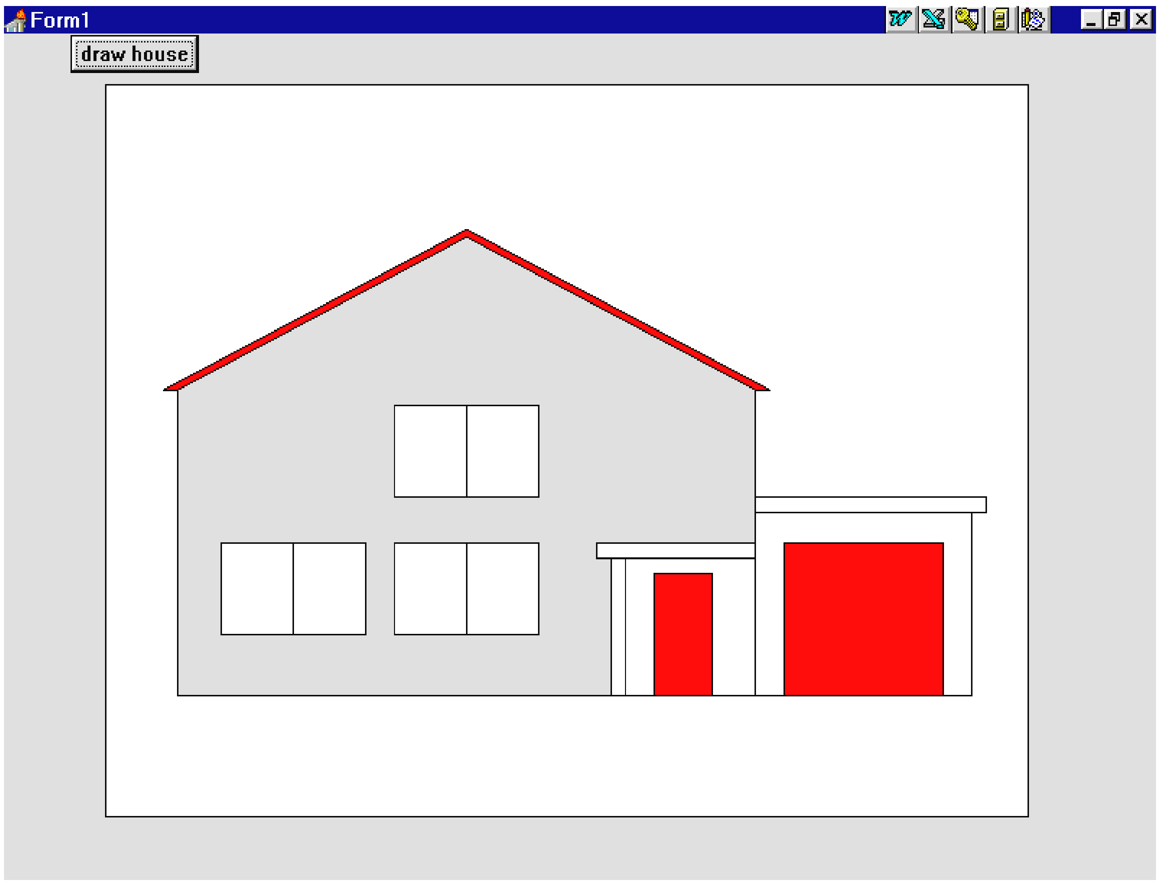
This will colour the garage door red:

- The **brush.color** command sets the fill colour to be red
- The **floodfill** command starts at the point (500 across, 350 down) and spreads colour in all directions until a black border is reached. You can use any coordinates which are within the area to be flood filled.

Other colours are available. The colour codes are listed below.

Standard colour codes for use with the brush.color command:

clBlack	clMaroon
clGreen	clOlive
clNavy	clPurple
clTeal	clGray
clSilver	clRed
clLime	clBlue
clFuchsia	clAqua
clWhite	



SUMMARY

In this chapter you have:

- Used the *shape* component to produce rectangles and a circle
- Selected the fill colour for the shapes
- Produced shapes with and without outlines
- Used the *label* component to display text
- Selected the font and size for the label text
- Seen how run time graphics make use of a screen coordinate system like graph paper
- Set up an image box with specified width and height coordinates
- Used **moveto**, **lineto** and **rectangle** commands to produce graphics
- Used **brush.color** and **floodfill** commands to fill areas of the picture with colour