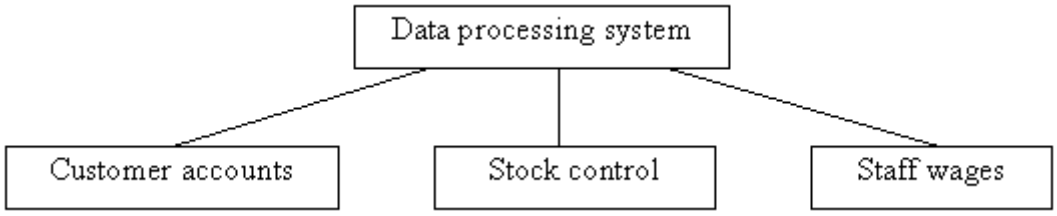
THREE

Multiple windows

Often in a Windows application it is convenient to have more than one window in use. This provides a natural way of breaking down a large and complex project into a number of simpler modules.

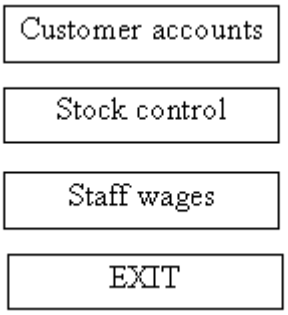
Consider a project to computerise the data processing needs of a business. The software might be required to carry out a number of different functions as shown below:



A diagram of this kind, which subdivides the overall project into a number of subsidiary modules, is known as a top-down structure diagram. In Delphi, each of the subsidiary modules would have its own program unit and screen form. Let's go some way towards setting up the structure for this project.

Set up a new sub-directory MULTI in your work area, open a new Delphi project and save it into the sub-directory.

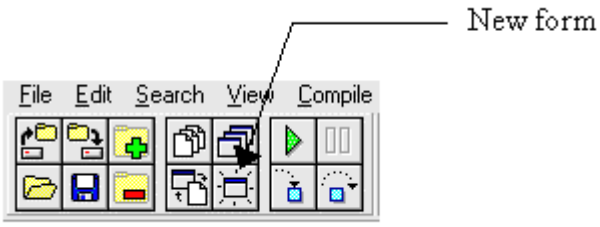
We will design Form1 to be a menu screen. Use the Object Inspector to set the WindowState to Maximized, and drag the grid to nearly fill the screen. Add four buttons and label them:



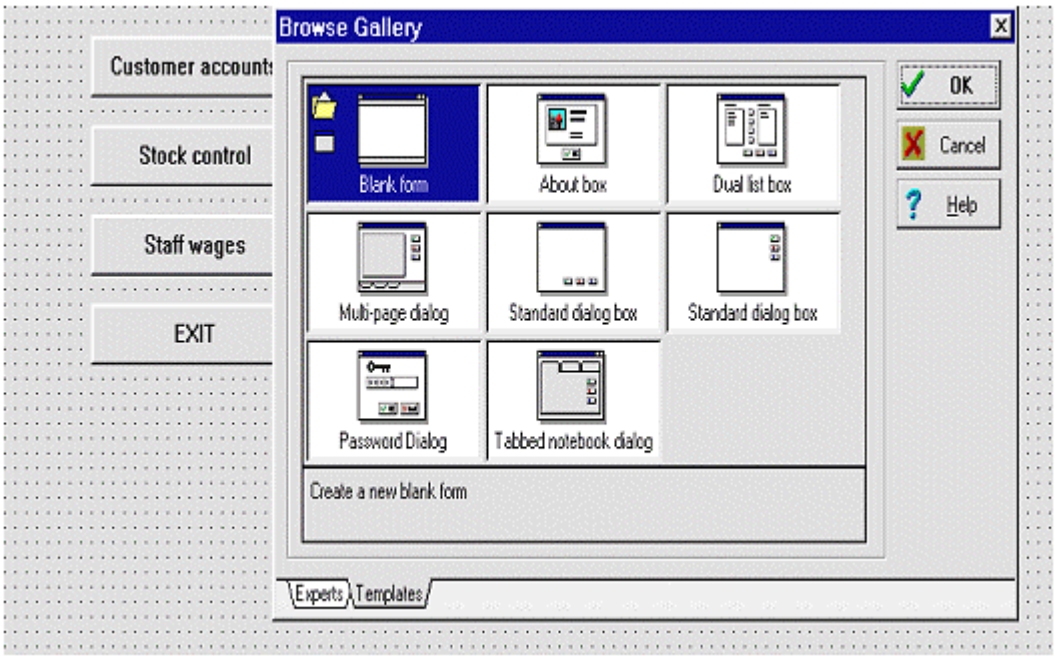
Double-click the EXIT button to create an event handler procedure, and add the halt command:

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  halt;
end;
```

Run the program to check that it exits correctly when the button is clicked.
We are now going to create another program unit and screen form which would run the customer accounts section of the completed data processing system. Click on the New form short cut button:

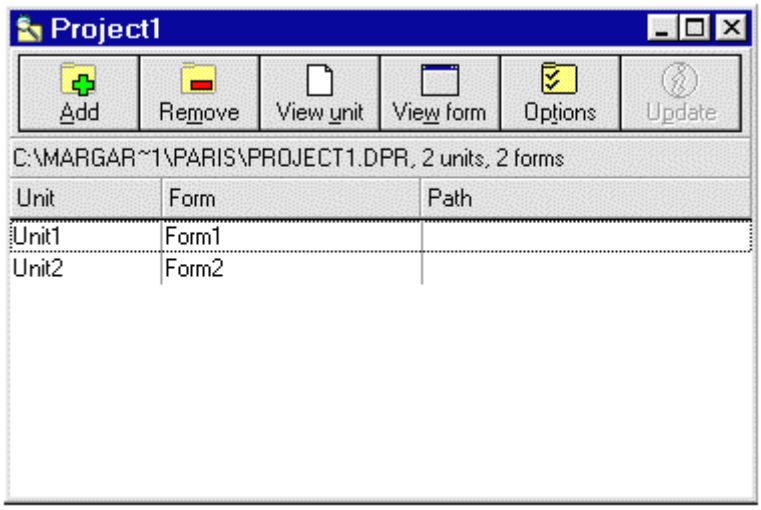


A selection window will appear. Choose the Blank form option and click the OK button:



Another grey dotted grid window will appear, this time labelled Form2. Delphi has also produced a program unit called Unit2 to operate and control it. Use the Save Project short cut button to save Unit2 into your project sub-directory.

In a large Delphi project there may be a dozen or more screen forms and their accompanying program units, so it is important to be able to locate the particular window you wish to work on. To do this, select the VIEW drop down menu, then click the PROJECT MANAGER option. A list will be displayed of all units and forms currently in the project. To select a particular window, click on the unit name then click either the **View unit** or **View form** button.



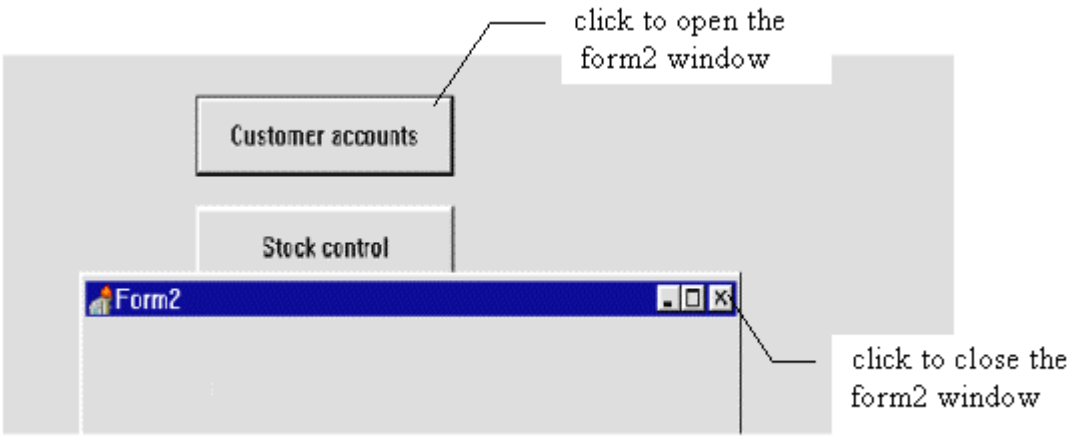
We will now add program instructions to link the units. Using the project manager table, bring the Form1 window to the front of the screen. Double-click the 'Customer accounts' button to create an event handler procedure. Add a line of program:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  form2.visible:=true;
end;
```

Scroll towards the top of the program to find the **Uses** line and add 'unit2' to the list of entries:

```
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls, Unit2;
```

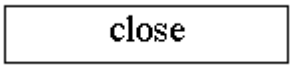
Go to the COMPILE drop down menu, but this time choose the **Build All** option. **Build All** is necessary for linking together the units of a multi-window application.
When the project has compiled successfuly, run the program.



Clicking the 'customer accounts' button will make the Form2 window appear. Clicking the cross in the corner of Form2 will close the window. Try this a few times then use the 'Exit' button to return to the Delphi editing screen.

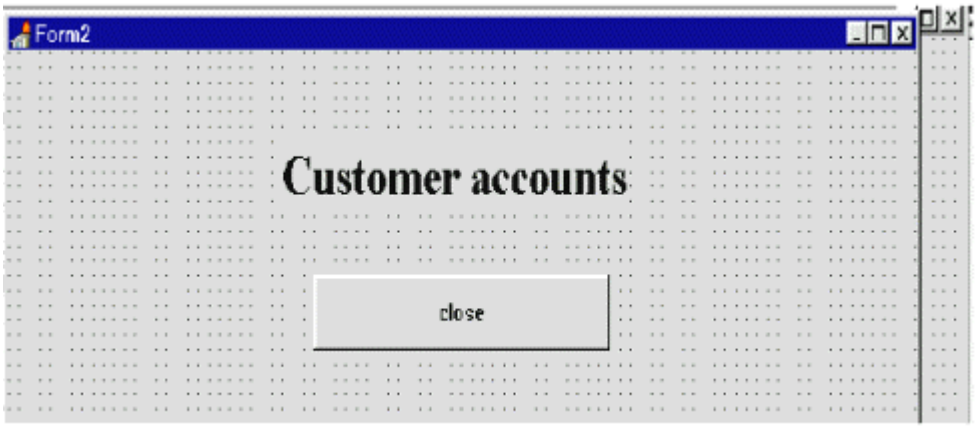
Use the **Program Manager** table to select the Form2 window. Press ENTER to bring up the Object Inspector, and set the **WindowState** to **Maximized**.

Drag the Form2 window larger until it nearly fills the screen. Add a button and give it the caption:



Double-click the button to create an event handler, and add a line of program:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  form2.visible:=false;
end;
```



Finish by adding a label with the caption 'Customer accounts' on the Form2 grid, using a large type face.

From the COMPILE drop down menu, choose **Build All** again. When the project is successfuly compiled, run it.

Pressing the 'Customer accounts' button will open up **Form2** to fill the screen. Pressing the 'close' button will close the Form2 window and return you to the **Form1** window. Try this a few times, then exit to the Delphi editing screen.

See if you can complete the module system by creating Forms and Program Units for the other two menu options: Stock control and Staff wages. In each case you will need to:

- add a new form
- save this in the project sub-directory
- create an event handler procedure for the button on Form1 and add a line of program to make the new form visible
- make an entry in the Form1 Uses list to include the new unit
- use the object inspector to maximize the new form
- add a button to the form. Create an event handler procedure with a command to close the form when the button is pressed
- give the the new form a label heading

Use COMPILE and **Build All**, then try out the completed menu system.

Paris Tourist information

We are now ready to tackle a substantial programming project in Delphi, putting into practice many of the ideas learned in the course so far:

Set up a sub-directory in your work area and name this PARIS. Start a new Delphi project and save into the PARIS sub-directory.

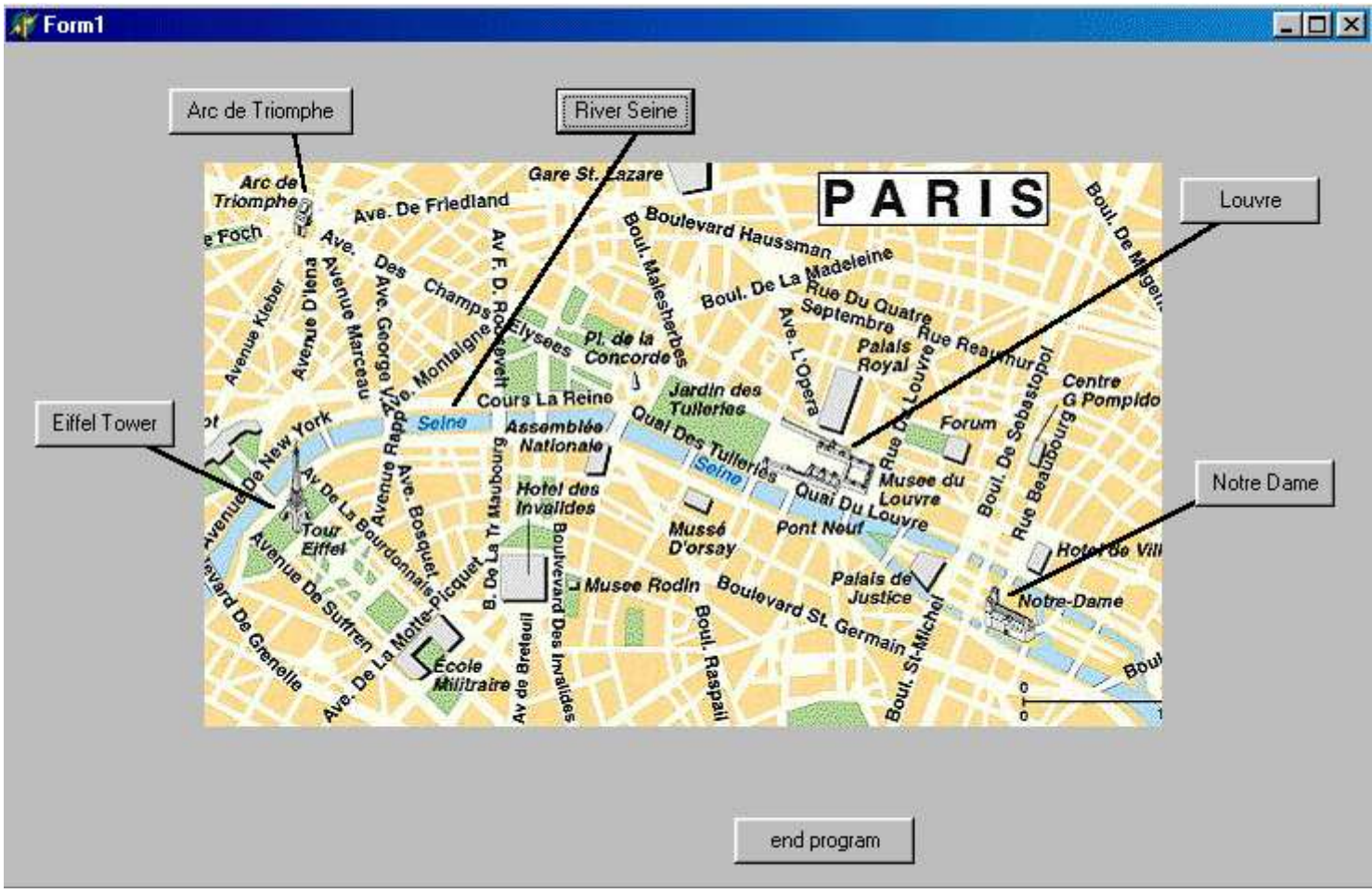
Set the **WindowState** to **Maximized**, and drag Form1 larger to nearly fill the screen.

Add an image box and load the bitmap file PARISMAP.BMP which is provided. Place buttons at the ends of the label lines on the map, and add captions:

Eiffel Tower | River Seine | Notre Dame

Arc de Triomphe | Louvre

Also add a button to | end program



Add an event handler procedure for the 'end program' button:

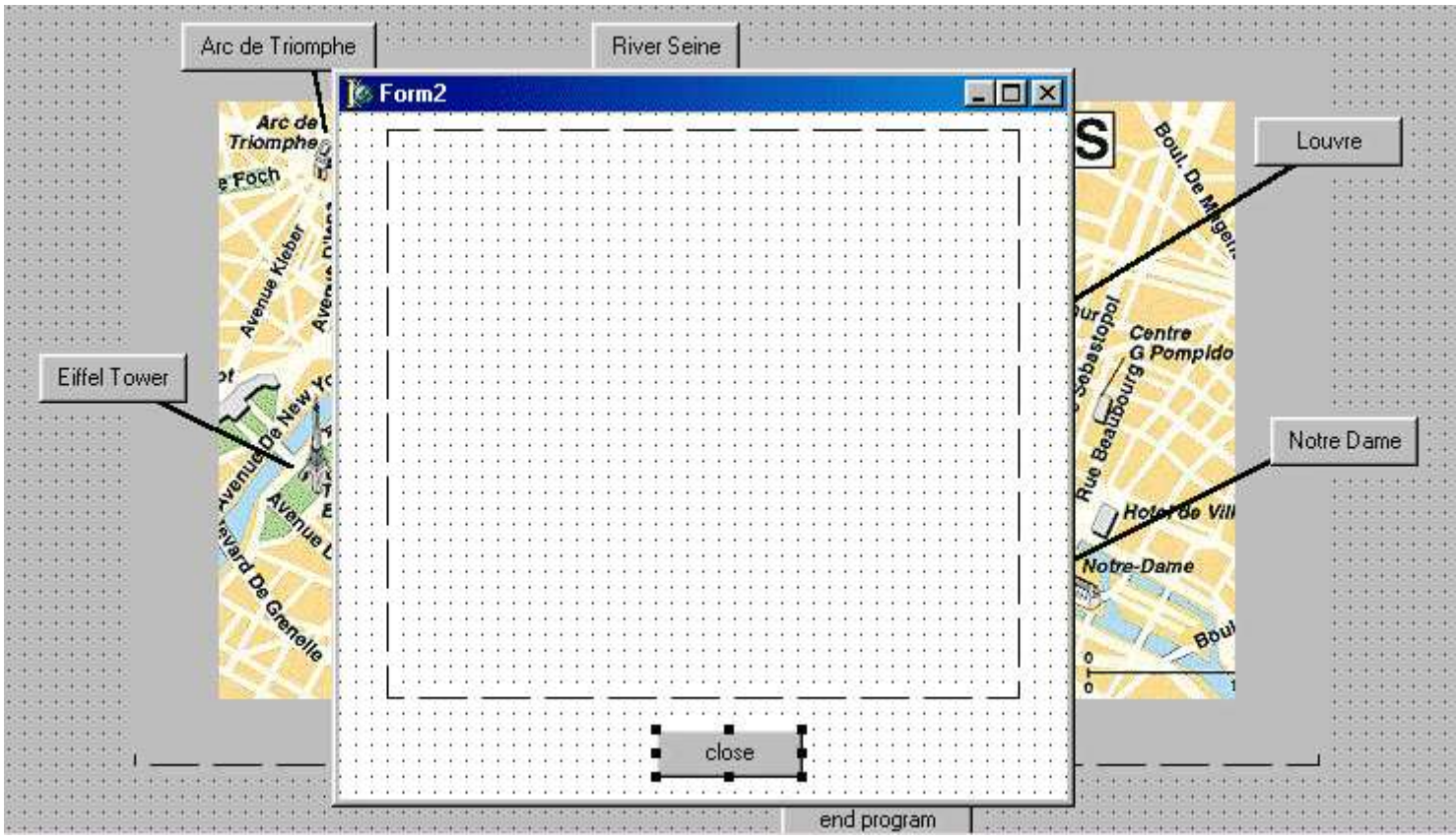
```
procedure TForm1.Button6Click(Sender: TObject);
begin
  halt;
end;
```

Compile and run the program to check that it exits correctly.

The purpose of the program will be to display photographs of places of interest in Paris when the user clicks the appropriate button. It would be convenient for the photographs to appear in a separate small window which can 'float' on top of the map screen. To set this up, first click the *New form* short cut button and select a **Blank Form**. A new window labelled **Form2** will appear.
We will now configure the **Form2** window ready to display the photographs.

Bring up the Object Inspector window by clicking on the Form2 grid and pressing ENTER. Find the **Color** property and set this to **clWhite**. The grid will now be displayed in white.

Adjust the size of the Form2 window to be a square, approximately the same height as the map image box on Form1. When the program is running, we do not wish the user to be able to adjust the size of this window - its size can be fixed by selecting the **BorderStyle** property on the Object Inspector, and setting this to **bsDialog**.



Now add an image box and button to Form2 as shown above. Give the button the caption:

close

Again using the Object Inspector for Form2, find the FormStyle property and set this to fsStayOnTop. This will ensure that the photograph window will 'float' on top of the map screen while the program is running.

Double-click the 'close' button to create an event handler. Add a line of program to close the window:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  form2.visible:=false;
end;
```

A series of photographs have been provided for you in bitmap format:

PARIS1.BMP	River Seine
PARIS2.BMP	Eiffel Tower
PARIS3.BMP	Louvre
PARIS4.BMP	Norte Dame
PARIS5.BMP	Arc de Triomphe

Use Windows Explorer to copy these into your PARIS sub-directory. The program will need to have access to these files while it is running.
Click on the dotted grid of Form1 to bring this to the front. Double-click the 'River Seine' button to create an event handler and add the lines of program:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  form2.visible:=true;
  form2.image1.picture.loadfromfile('paris1.bmp');
end;
```

The purpose of these instructions are to make the Form2 window visible on top of the map, and to load the photograph of the River Seine into the image box on the form.
Scroll towards the top of the program and add Unit2 to the Uses line:

```
uses
 SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
 Controls, Forms, Dialogs, StdCtrls, ExtCtrls, unit2;
```

Use the Build All option on the COMPILE drop down menu, then run the program.

The map screen will first be displayed. Clicking the 'River Seine' button should open the Form2 window and display the photograph. Notice that the Form2 window can be moved around on top of the map by dragging the top blue border of the window. Click the 'close' button to hide Form2.

Exit from the program and return to the Delphi editor. Bring Form1 to the front and double-click on the 'Eiffel Tower' button. Add lines of program to the event handler:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  form2.visible:=true;
  form2.image1.picture.loadfromfile('paris2.bmp');
end;
```

This is very similar to the event handler for the 'River Seine' button, except that this time the file name is 'paris2.bmp'. Use Build All, then run the program to check that a picture of the Eiffel Tower is displayed.

Return to the editing screen and see if you can set up the event handler procedures to display the other three photographs. Save the finished project in the PARIS sub-directory.