



**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**Stock Property by Using Delphi**

**Graduation Project**

**COM 400**

**Student: Seda ONHAN (20032905)**

**Supervisor: Assist. Prof. Dr. Imanov ELBRUS**

**Nicosia – 2008**





## ACKNOWLEDGEMENT

*First of all, I would like give my special thanks to my supervisor Assist. Prof. Dr. Imanov ELBRUS. He helped and supported me to complete my project by any means of necessary. In addition to this he never doubted about me, he always believed in me that I will fulfill and succeed on my project. I am glad to that I did not disappoint him.*

*Furthermore, I want to give my special thanks and best regards to my parents. They were always kind and patient to me. I wouldn't be here without their endless support.*

*Finally, I want to give my special thanks to my friends whose are Cemal Kavalcioğlu, Selman Oğuzhan ESER. They are supported and helped me to complete my project. I am very happy to have such friends.*

## **ABSTRACT**

The aim of this Project is to record the stock device for any Properties Company. The program was prepared by using Delphi 7 programming and using Paradox7. Delphi is a programming language that can be used with Paradox7.

This project consists of many different pages but most of them depended each other. Initially, SIGN IN form comes to screen. Afterwards the Main menu of Properties Company comes to screen. After Main Menu you are going to see the main form that contains 15 main menus.

## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENT .....</b>	<b>I</b>
<b>ABSTRACT.....</b>	<b>II</b>
<b>TABLE OF CONTENTS .....</b>	<b>III</b>
<b>INTRODUCTION .....</b>	<b>1</b>

### CHAPTER ONE : BASIC CONCEPT OF DELPHI

1.1 Introduction to Delphi.....	2
1.2 What is Delphi? .....	2
1.2.1 Delphi Compilers .....	2
1.2.2 What kind of programming can you do with Delphi? .....	3
1.2.3 History of Delphi .....	4
1.2.4 Advantages & Disadvantages Delphi .....	6
1.3 Delphi 6 Editions .....	7
1.3.1 Delphi 6 Architect.....	7
1.3.2 Installation Delphi 6.....	8
1.4 A Tour of the Environment.....	10
1.4.1 Running Delphi for the First Time .....	10
1.4.2 The Delphi IDE.....	11
1.4.3 The Menus & Toolbar.....	12
1.4.4 The Component Palette.....	12
1.4.5 The Code Editor.....	13
1.4.6 The Object Inspector.....	14
1.4.7 The Object TreeView.....	15
1.4.8 Class Completion.....	16
1.4.9 Debugging applications .....	17
1.4.10 Exploring Databases .....	18
1.4.11 Templates and the Object Repository .....	19
1.5 Programming with Delphi .....	20
1.5.1 Starting a New Application.....	20
1.5.2 Setting Property Values .....	21
1.5.3 Adding objects to the form .....	22
1.5.4 Add a Table and a StatusBar to the Form.....	22
1.5.5 Connecting to a Database .....	24

### CHAPTER TWO : THE RAVE REPORTING

2.1 Project Tree.....	28
2.2 Design Tools .....	29
2.3 Reuse and Maintenance Tools .....	32
2.4 Standard Components .....	34
2.5 Drawing Components .....	35
2.6 Reporting Components .....	35
2.7 Barcode Components.....	39
2.8 Anchors .....	39



2.9	Code Based Reports .....	40
2.9.1	Simple Code Base Report .....	40
2.9.2	Tabular Code Based Report .....	41
2.9.3	Graphical Code Based Report.....	43
2.10	Visually Designed Reports .....	45
2.10.1	The Visual Designer .....	45
2.10.2	Interacting with the Project.....	48
2.11	Data Aware Reports.....	55
2.11.1	The Database Connection .....	55
2.11.2	The Driver Data View.....	55
2.11.3	Regions and Bands.....	58
2.11.4	Adding Fields.....	60
2.11.5	Adding the Report to Your Project .....	60

## CHAPTER THREE : STOCK PROPERTY BY USING DELPHI

3.1	Database Connection Screen .....	61
3.2	Main Menu.....	63
3.3	House to Let Menu .....	64
3.3.1	House to Let Organize Form .....	64
3.3.2	House to Let Search Form .....	65
3.3.3	House to Let Report Form .....	68
3.4	House for Sale Menu .....	69
3.4.1	House for Sale Organize Form .....	69
3.4.2	House for Sale Search Form .....	70
3.4.3	Hose for Sale Report Form .....	73
3.5	Shop to Let Menu .....	74
3.5.1	Shop to Let Organize Form .....	74
3.5.2	Shop to Let Search Form .....	75
3.5.3	Shop to Let Report Form .....	78
3.6	Shop for Sale Menu .....	79
3.6.1	Shop for Sale Organize Form .....	79
3.6.2	Shop for Sale Search Form .....	80
3.6.3	Shop for Sale Report Form .....	83
3.7	Plot to Let Menu .....	84
3.7.1	Plot to Let Organize Form .....	84
3.7.2	Plot to Let Search Form .....	85
3.7.3	Plot to Let Report Form .....	88
3.8	Garden for Sale Menu.....	89
3.8.1	Garden for Sale Organize Form.....	89
3.8.2	Garden for Sale Search Form.....	90
3.8.3	Garden for Sale Report Form.....	93
3.9	Building for Sale Menu.....	94
3.9.1	Building for Sale Organize Form.....	94
3.9.2	Buildig for Sale Search Form .....	95
3.9.3	Building for Sale Report Form .....	98
3.10	Farm for Sale Menu .....	99
3.10.1	Farm for Sale Organize Form .....	99

3.10.2	Farm for Sale Search Form .....	100
3.10.3	Farm for Sale Report Form .....	103
3.11	Villa for Sale Menu .....	104
3.11.1	Villa for Sale Organize Form .....	104
3.11.2	Villa for Sale Search Form .....	105
3.11.3	Villa for Sale Report Form .....	108
3.12	Field for Sale Menu .....	109
3.12.1	Field for Sale Organize Form .....	109
3.12.2	Filed for Sale Search Form .....	110
3.12.3	Field for Sale Report Form .....	113
3.13	Flier Print Menu .....	114
3.13.1	Flier Print Organize Form .....	114
3.13.2	House to Let Advertisements Form .....	115
3.13.3	Villa for Sale Advertisements Form .....	116
3.13.4	Shop to Let Advertisements Form .....	117
3.13.5	Plot for Sale Advertisements Form .....	118
3.13.6	House for Sale Advertisements Form .....	119
3.13.7	Field for Sale Advertisements Form .....	120
3.13.8	Shop for Sale Advertisements Form .....	121
3.13.9	Garden for Sale Advertisements Form .....	122
3.13.10	Building for Sale Advertisements Form .....	123
3.13.11	Farm for Sale Advertisements Form .....	124
3.14	User Register Menu .....	125
3.15	About Menu .....	126
3.16	Informations Menu .....	127
3.17	Exit Menu .....	128
<b>CONCLUSION</b> .....		129
<b>REFERENCES</b> .....		130
<b>APPENDIX</b> .....		140

## INTRODUCTION

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

The project consists of the introduction, three chapters, and conclusion.

- Chapter one describes Basic Concept of Delphi.
- Chapter two describes the database that uses Delphi programming language.
- Chapter three explains Stock Property by Using Delphi.

## **CHAPTER ONE**

### **1 BASIC CONCEPT OF DELPHI**

#### **1.1 Introduction to Delphi**

Although I am not the most experienced or knowledgeable person on the forums I thought it was time to write a good introductory article for Delphi

#### **1.2 What is Delphi?**

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

For the purposes of this series I will be using Delphi 6. Delphi 6 provides all the tools you need to develop test and deploy Windows applications, including a large number of so-called reusable components.

Borland Delphi provides a cross platform solution when used with Borland Kylix – Borland's RAD tool for the Linux platform.

##### **1.2.1 Delphi Compilers**

There are two types compiler for Delphi

- Turbo Delphi: Free industrial strength Delphi RAD (Rapid Application Development) environment and compiler for Windows. It comes with 200+ components and its own Visual Component Framework.



- Turbo Delphi for .NET: Free industrial strength Delphi application development environment and compiler for the Microsoft .NET platform.

### **1.2.2 What kind of programming can you do with Delphi?**

The simple answer is “more or less anything”. Because the code is compiled, it runs quickly, and is therefore suitable for writing more or less any program that you would consider a candidate for the Windows operating system.

You probably won't be using it to write embedded systems for washing machines, toasters or fuel injection systems, but for more or less anything else, it can be used (and the chances are that probably someone somewhere has!)

Some projects to which Delphi is suited:

- Simple, single user database applications
- Intermediate multi-user database applications
- Large scale multi-tier, multi-user database applications
- Internet applications
- Graphics Applications
- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools
- Communications tools using the Internet, Telephone or LAN
- Web based applications

This is not intended to be an exhaustive list, more an indication of the depth and breadth of Delphi's applicability. Because it is possible to access any and all of the Windows API, and because if all else fails, Delphi will allow you to drop a few lines of assembler code directly into your ordinary Pascal instructions, it is possible to do more or less anything. Delphi can also be used to write Dynamically Linked Libraries (DLLs) and can call out to DLLs written in other programming languages without difficulty.

Because Delphi is based on the concept of self contained Components (elements of code that can be dropped directly on to a form in your application, and exist in object form, performing their function until they are no longer required), it is possible to build applications very rapidly. Because Delphi has been available for quite some time, the number of pre-written components has been increasing to the point that now there is a component to do more or less anything you can imagine. The job of the programmer has become one of gluing together appropriate components with code that operates them as required.

### **1.2.3 History of Delphi**

Delphi was one of the first of what came to be known as "RAD" tools, for Rapid Application Development, when released in 1995 for the 16-bit Windows 3.1. Delphi 2, released a year later, supported 32-bit Windows environments, and a C++ variant, C++ Builder, followed a few years after.

The chief architect behind Delphi, and its predecessor Turbo Pascal, was Anders Hejlsberg until he was headhunted in 1996 by Microsoft, where he worked on Visual J++ and subsequently became the chief designer of C Sharp programming language C# and a key participant in the creation of the Microsoft .NET Framework.

In 2001 a Linux version known as Kylix programming tool Kylix became available. However, due to low quality and subsequent lack of interest, Kylix was abandoned after version 3.

Support for Linux and Windows cross platform development (through Kylix and the CLX component library) was added in 2002 with the release of Delphi 6.

Delphi 8, released December 2003, was a .NET -only release that allowed developers to compile Delphi Object Pascal code into .NET Microsoft Intermediate Language MSIL. It was also significant in that it changed its IDE for the first time, from the multiple-floating-window-on-desktop style IDE to a look and feel similar to Microsoft's Visual Studio.NET.



Although Borland fulfilled one of the biggest requests from developers (.NET support), it was criticized both for making it available too late, when a lot of former Delphi developers had already moved to C#, and for focusing so much on backward compatibility that it was not very easy to write new code in Delphi. Delphi 8 also lacked significant high-level features of the c sharp, C# language, as well as many of the more appealing features of Microsoft's Visual Studio IDE. (There were also concerns about the future of Delphi Win32 development. Because Delphi 8 did not support Win32, Delphi 7.1 was included in the Delphi 8 package.)

The next version, Delphi 2005 (Delphi 9), included the Win32 and .NET development in a single IDE, reiterating Borland's commitment to Win32 developers. Delphi 2005 includes design-time manipulation of live data from a database. It also includes an improved IDE and added a "for ... in" statement (like C#'s for each) to the language. However, it was criticized by some for its bugs; both Delphi 8 and Delphi 2005 had stability problems when shipped, which were only partially resolved in service packs.

In late 2005, Delphi 2006 was released and federated development of C# and Delphi.NET, Delphi Win32 and C++ into a single IDE. It was much more stable than Delphi 8 or Delphi 2005 when shipped, and improved even more after the service packs and several hot fixes.

On February 8, 2006, Borland announced that it was looking for a buyer for its IDE and database line of products, which include Delphi, to concentrate on its Application Lifecycle Management ALM line. The news met with voluble optimism from the remaining Delphi users.

On September 6, 2006, The Developer Tools Group (the working name of the not yet spun off company) of Borland Software Corporation released single language versions of Borland Developer Studio, bringing back the popular "Turbo" moniker. The Turbo product set includes Turbo Delphi for Win32, Turbo Delphi for .NET, Turbo C++, and Turbo C#. Each version is available in two editions: "Explorer" a free downloadable version and "Professional" a relatively cheap (US\$399) version which

opens access to thousands of third-party components. Unlike earlier “Personal” editions of Delphi, new “Explorer” editions can be used for commercial development.

On November 14, 2006, Borland announced the cancellation of the sale of its Development tools; instead of that it would spin them off into an independent company named “CodeGear”

#### **1.2.4 Advantages & Disadvantages Delphi**

Delphi exhibits the following advantages:

- Rapid Application Development (RAD)
- Based on a well-designed language – high-level and strongly typed, with low-level escapes for experts
- A large community on Usenet and the World Wide Web (e.g. [news://newsgroups.borland.com](mailto:news://newsgroups.borland.com) and Borland’s web access to Delphi)
- Can compile to a single executable, simplifying distribution and reducing DLL versioning issues
- Many VCL and third-party components (usually available with full source code) and tools (documentation, debug tools, etc.)
- Quick optimizing compiler and ability to use assembler code
- Multiple platform native code from the same source code
- High level of source compatibility between versions
- Cross Kylix – a third-party toolkit which allows you to compile native Kylix/Linux applications from inside the Windows Delphi IDE, hence easily enabling dual-platform development and deployment
- Cross FBC – a sister project to Cross Kylix, which enables you to cross-compile your Windows Delphi applications to multi-platform targets – supported by the Free Pascal compiler – without ever leaving the Delphi IDE
- Class helpers to bridge functionality available natively in the Delphi RTL, but not available in a new platform supported by Delphi
- The language’s object orientation features only class- and interface-based Polymorphism in object-oriented programming polymorphism

Disadvantages:

- Limited cross-platform capability for Delphi itself. Compatibles provide more architecture/OS combinations
- Access to platform and third party libraries require header files to be translated to Pascal. This creates delays and introduces the possibilities of errors in translation.
- There are fewer published books on Delphi than on other popular programming languages such as C++ and C#
- A reluctance to break any code has lead to some convoluted language design choices, and orthogonally and predictability have suffered

### **1.3 Delphi 6 Editions**

There are 3 editions in Delphi 6:

- Delphi Personal – makes learning to develop non-commercial Windows applications fast and fun. Delphi 6 Personal makes learning Windows development easy with drag-and-drop visual programming.
- Delphi Professional – adds the tools necessary to create applications with the latest Windows® ME/2000 look-and-feel. Dramatically enhance functionality with minimal code using the power and flexibility of SOAP and XML to easily integrate Web Services into client-side applications.
- Delphi Enterprise – includes additional tools, extensive options for Internet. Delphi 6 makes next-generation e-business development with Web Services a snap.

This Program will concentrate on the Enterprise edition.

#### **1.3.1 Delphi 6 Architect**

Delphi 6 Architect is designed for professional enterprise developers who need to adapt quickly to changing business rules and manage sophisticated applications that synchronize with multiple database schemas. Delphi 2006 Architect includes an advanced ECO III framework that allows developers to rapidly deploy scalable external facing Web applications with executable state diagrams, object-relational mapping, and transparent persistence.



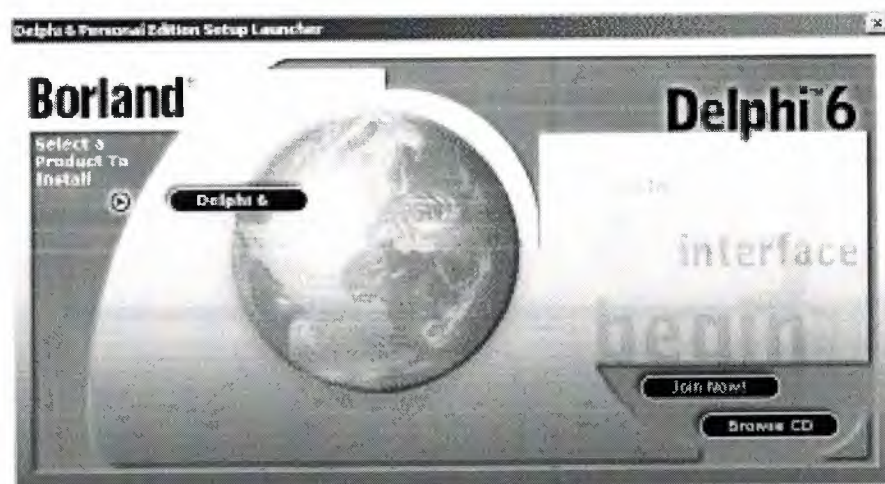
Delphi 6 Architect includes all of the capabilities of the Enterprise edition, and includes the complete ECO III framework, including new support for ECO State Machines powered by State Chart visual diagrams, and simultaneous persistence to multiple and mixed database servers.

- State Chart Diagrams
- Executable ECO State Machines
- Multi- and Mixed- ECO database support

### 1.3.2 Installation Delphi 6

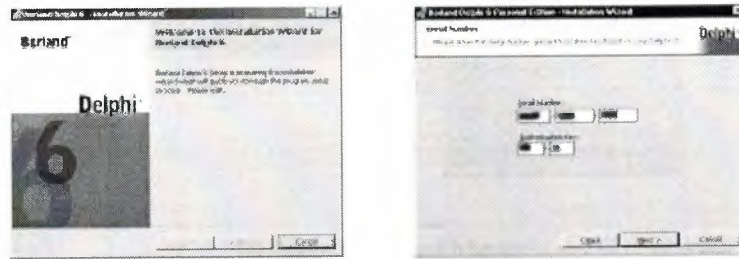
To install Delphi 6 Enterprise, run INSTALL.EXE (default location C:\Program Files\Borland Delphi) and follow the installation instructions.

We are prompted to select a product to install; you only have one choice “Delphi 6”:



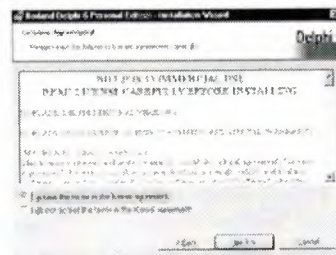
**Figure 1.1** The Select Page For Start Installation

While the setup runs, you'll need to enter your serial number and the authorization key (the two you got from inside a CdRom driver).



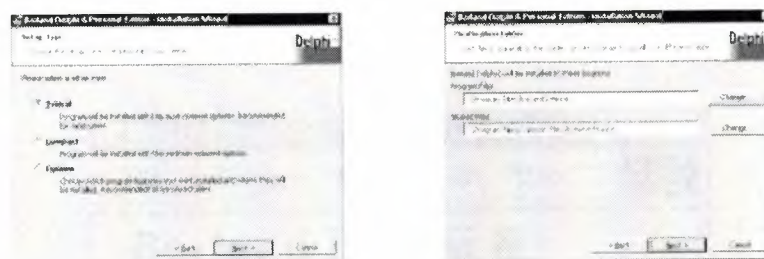
**Figure 1.2** Serial Number And Authorization Screen

Later, the License Agreement screen wills popup:



**Figure 1.3** License Agreement Screen

After that, you have to pick the Setup Type, choose Typical. This way Delphi 6 Enterprise will be installed with the most common options. The next screen prompts you to choose the Destination folder.



**Figure 1.4** SetUp Type and Destination Folder Screen

At the end of the installation process, the set-up program will create a sub menu in the Programs section of the Start menu, leading to the main Delphi 6 Enterprise program plus some additional tools.



**Figure 1.5** Start Menu

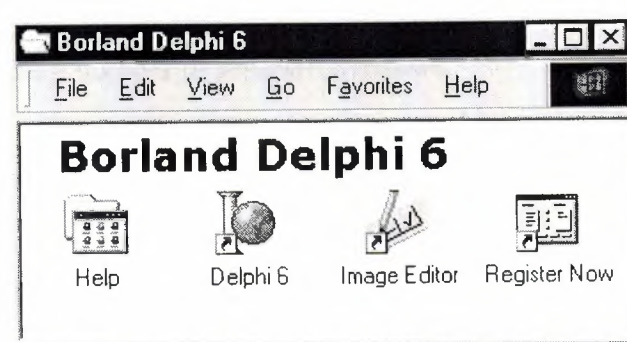
## 1.4 A Tour of the Environment

This chapter explains how to start Delphi and gives you a quick tour of the main parts and tools of the Integrated Development Environment (IDE)

### 1.4.1 Running Delphi for the First Time

You can start Delphi in a similar way to most other Windows applications:

- Choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu
- Choose Run from the Windows Start menu and type Delphi32
- Double-click Delphi32.exe in the \$(DELPHI)\Bin folder. Where \$(DELPHI) is a folder where Delphi was installed. The default is C:\Program Files\Borland\Delphi6.
- Double-click the Delphi icon on the Desktop (if you've created a shortcut)



**Figure 1.6** Borland Delphi 6 Folder



### 1.4.2 The Delphi IDE

As explained before, one of the ways to start Delphi is to choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu.

When Delphi starts (it could even take one full minute to start – depending on your hardware performance) you are presented with the IDE: the user interface where you can design, compile and debug your Delphi projects.

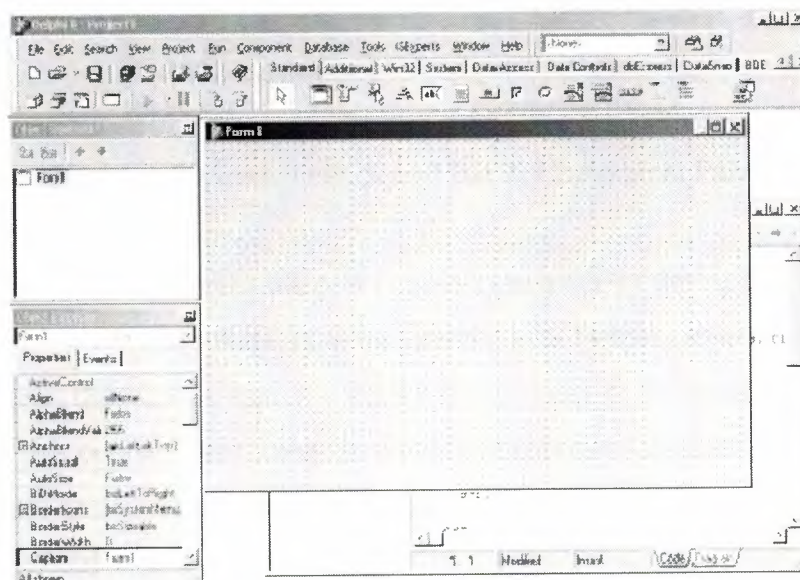


Figure 1.7 IDE

Like most other development tools (and unlike other Windows applications), Delphi IDE comprises a number of separate windows.

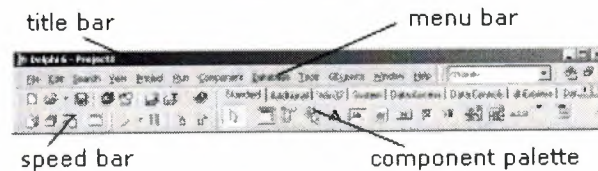
Some of the facilities that are included in the “Integrated Development Environment” (IDE) are listed below:

- A syntax sensitive program file editor
- A rapid optimizing compiler
- Built in debugging /tracing facilities
- A visual interface developer
- Syntax sensitive help files
- Database creation and editing tools

- Image/Icon/Cursor creation / editing tools
- Version Control CASE tools

### 1.4.3 The Menus & Toolbar

The main window, positioned on the top of the screen, contains the main menu, toolbar and Component palette.



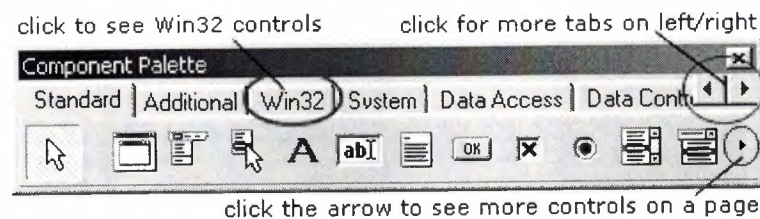
**Figure 1.8** Menu, Title, Speed Bar & Component Palette

The title bar of the main window contains the name of the current project (you'll see in some of the future chapters what exactly is a Delphi project). The menu bar includes a dozen drop-down menus – we'll explain many of the options in these menus later through this course. The toolbar provides a number of shortcuts to most frequently used operations and commands – such as running a project, or adding a new form to a project. To find out what particular button does, point your mouse “over” the button and wait for the tool tip. As you can see from the tool tip (for example, point to [Toggle Form/Unit]), many tool buttons have keyboard shortcuts ([F12]).

The menus and toolbars are freely customizable. I suggest you to leave the default arrangement while working through the chapters of this course.

### 1.4.4 The Component Palette

You are probably familiar with the fact that any window in a standard Windows application contains a number of different (visible or not to the end user) objects, like: buttons, text boxes, radio buttons, check boxes etc. In Delphi programming terminology such objects are called controls (or components). Components are the building blocks of every Delphi application. To place a component on a window you drag it from the component palette. Each component has specific attributes that enable you to control your application at design and run time.



**Figure 1.9** Component Palates

Depending on the version of Delphi (assumed Delphi 6 Personal through this course), you start with more than 85 components at your disposal – you can even add more components later (those that you create or from a third party component vendor).

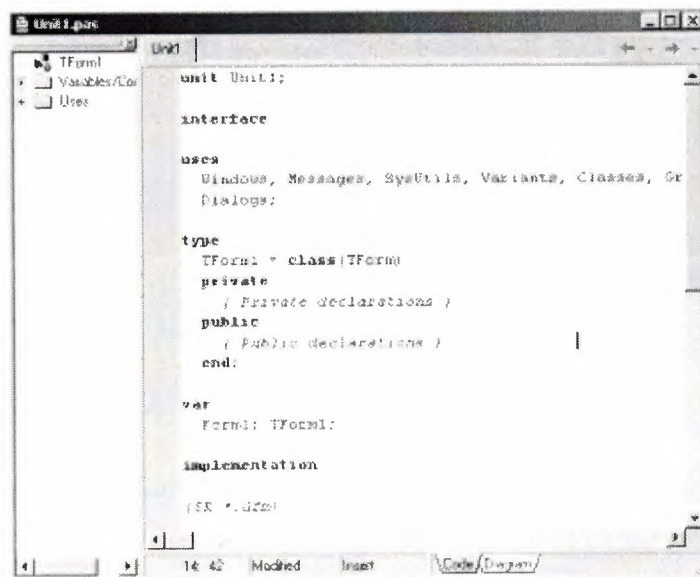
The components on the Component Palette are grouped according to the function they perform. Each page tab in the Component palette displays a group of icons representing the components you can use to design your application interface. For example, the Standard and Additional pages include controls such as an edit box, a button or a scroll box.

To see all components on a particular page (for example on the Win32 page) you simply click the tab name on the top of the palette. If a component palette lists more components that can be displayed on a page an arrow will appear on a far right side of the page allowing you to click it to scroll right. If a component palette has more tabs (pages) that can be displayed, more tabs can be displayed by clicking on the arrow buttons on the right-hand side.

#### **1.4.5 The Code Editor**

Each time you start Delphi, a new project is created that consists of one \*empty\* window. A typical Delphi application, in most cases, will contain more than one window – those windows are referred to as forms.

In our case this form has a name, it is called Form1. This form can be renamed, resized and moved, it has a caption and the three standard buttons which are minimize, maximize and close. As you can see a Delphi form is a regular Windows window



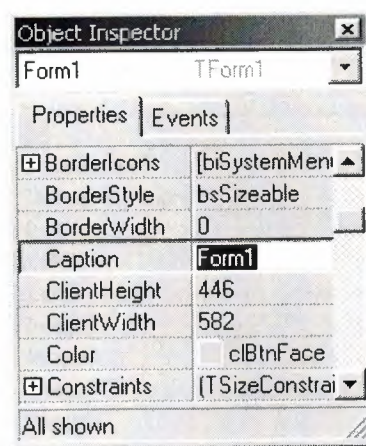
**Figure 1.10** Code Editor Window

If the Form1 is the active window and you press [F12], the Code Editor window will be placed on top. As you design user interface of your application, Delphi automatically generates the underlying Object Pascal code. More lines will be added to this window as you add your own code that drives your application. This window displays code for the current form (Form1); the text is stored in a (so-called) unit – Unit1. You can open multiple files in the Code Editor. Each file opens on a new page of the Code editor, and each page is represented by a tab at the top of the window.

#### **1.4.6 The Object Inspector**

Each component and each form has a set of properties – such as color, size, position, caption – that can be modified in the Delphi IDE or in your code, and a collection of events – such as a mouse click, keypress, or component activation – for which you can specify some additional behavior. The Object Inspector displays the properties and events (note the two tabs) for the selected component and allows you to change the property value or select the response to some event.





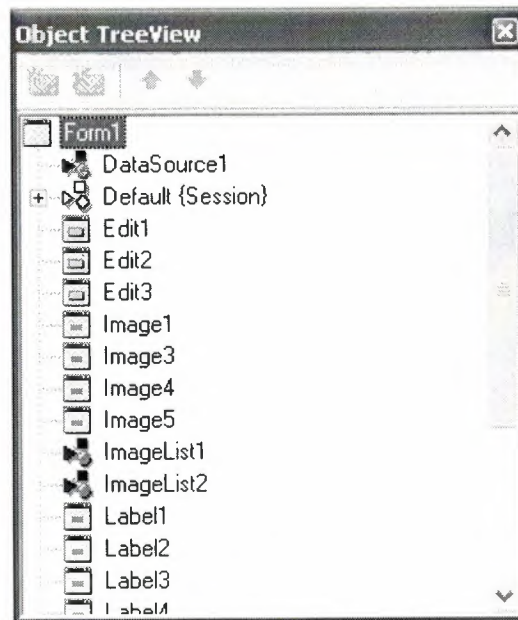
**Figure 1.11** Object Inspector

For example, each form has a Caption (the text that appears on it's title bar). To change the captions of Form1 first activate the form by clicking on it. In the Object Inspector find the property Caption (in the left column), note that it has the 'Form1' value (in the right column). To change the captions of the form simply type the new text value, like 'My Form' (without the single quotes). When you press [Enter] the caption of the form will change to My Form.

Note that some properties can be changed more simply, the position of the form on the screen can be set by entering the value for the Left and Top properties – or the form can be simply dragged to the desired location.

#### **1.4.7 The Object TreeView**

Above the Object Inspector you should see the Object TreeView window. For the moment its display is pretty simple. As you add components to the form, you'll see that it displays a component's parent-child relationships in a tree diagram. One of the great features of the Object TreeView is the ability to drag and drop components in order to change a component container without losing connections with other components.



**Figure 1.12** Object Tree View

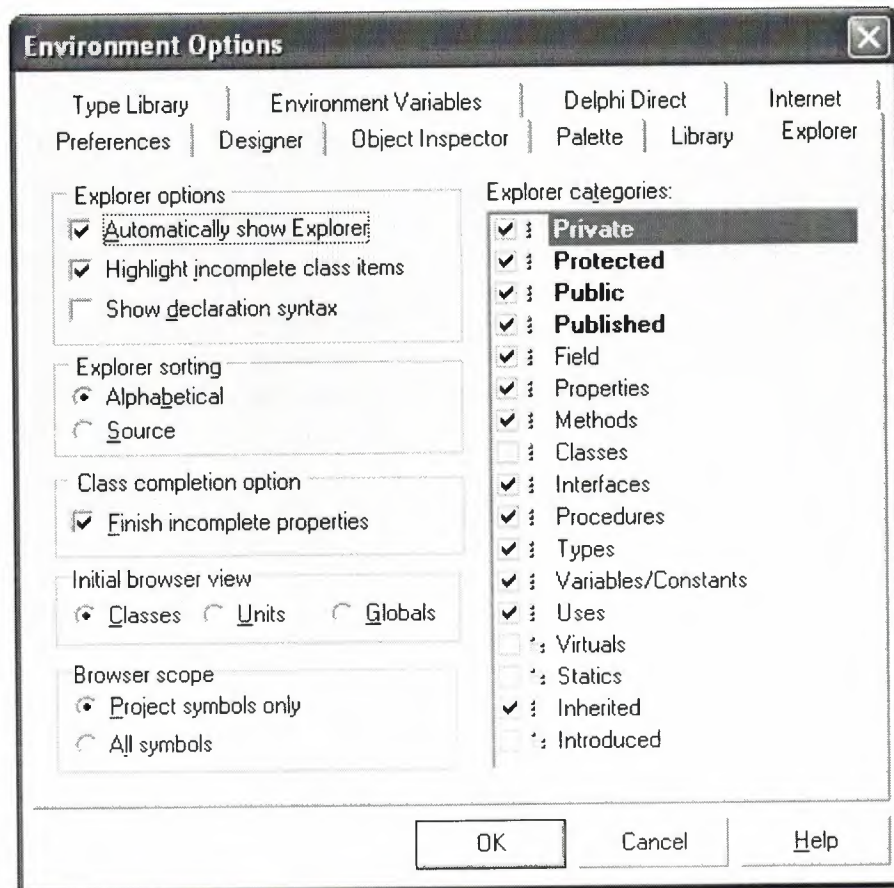
The Object TreeView, Object Inspector and the Form Designer (the Form1 window) work cooperatively. If you have an object on a form (we have not placed any yet) and click it, its properties and events are displayed in the Object Inspector and the component becomes focused in the Object TreeView.

#### **1.4.8 Class Completion**

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press Ctrl+Shift+C, or right-click and select Complete Class at Cursor. Delphi automatically adds private read and write specifies to the declarations for any properties that require them, and then creates skeleton code for all the class's methods. You can also use Class Completion to fill in class declarations for methods you've already implemented.

To configure Class Completion, choose Tools | Environment Options and click the Explorer tab.

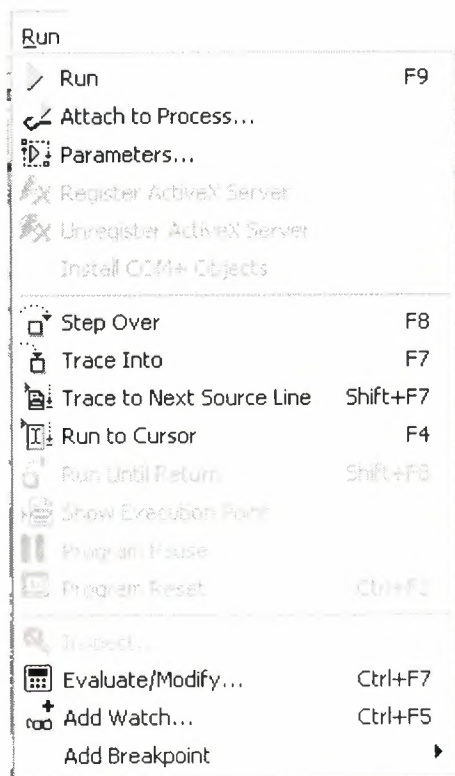




**Fig.1.13** Class Completion

#### 1.4.9 Debugging applications

The IDE includes an integrated debugger that helps you locate and fix errors in your code. The debugger lets you control program execution, watch variables, and modify data values while your application is running. You can step through your code line by line, examining the state of the program at each breakpoint.



**Figure1.14 Run**

To use the debugger, you must compile your program with debug information. Choose Project | Options, select the Compiler page, and check Debug Information. Then you can begin a debugging session by running the program from the IDE. To set debugger options, choose Tools | Debugger Options.

Many debugging windows are available, including Breakpoints, Call Stack, Watches, Local Variables, Threads, Modules, CPU, and Event Log. Display them by choosing View | Debug Windows. To learn how to combine debugging windows for more convenient use, see “Docking tool windows”.

#### **1.4.10 Exploring Databases**

The SQL Explorer (or Database Explorer in some editions of Delphi) lets you work directly with a remote database server during application development. For example, you can create, delete, or restructure tables, and you can import constraints while you are developing a database application.

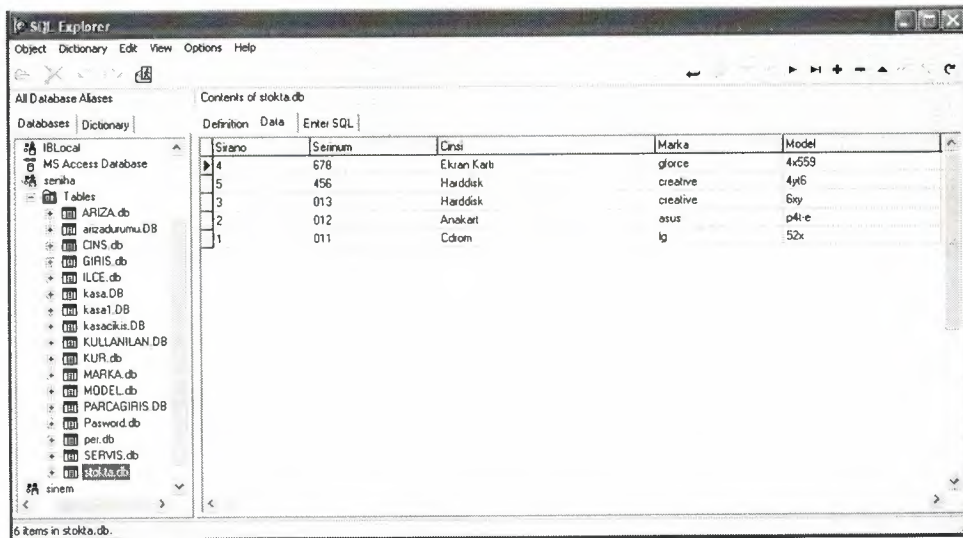


Figure 1.15 SQL Explorer

#### 1.4.11 Templates and the Object Repository

The Object Repository contains forms, dialog boxes, data modules, wizards, DLLs, sample applications, and other items that can simplify development. Choose File | New to display the New Items dialog when you begin a project. Check the Repository to see if it contains an object that resembles one you want to create.

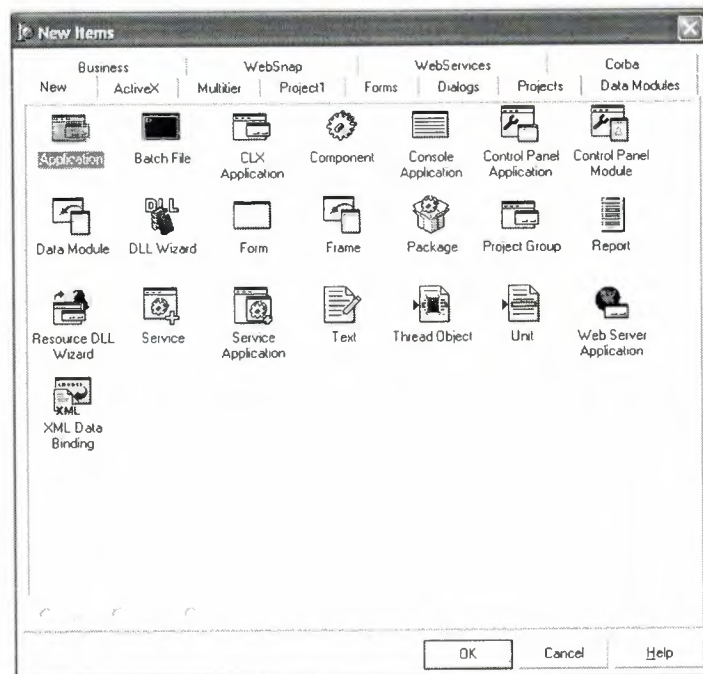


Figure 1.16 New Item

You can add your own objects to the Repository to facilitate reusing them and sharing them with other developers. Reusing objects lets you build families of applications with common user interfaces and functionality; building on an existing foundation also reduces development time and improves quality. The Object Repository provides a central location for tools that members of a development team can access over a network.

## **1.5 Programming with Delphi**

The following section provides an overview of software development with Delphi.

### **1.5.1 Starting a New Application**

Before beginning a new application, create a folder to hold the source files.

1. Create a folder in the Projects directory off the main Delphi directory.
2. Open a new project.

Each application is represented by a project. When you start Delphi, it opens a blank project by default. If another project is already open, choose File | New Application to create a new project. When you open a new project, Delphi automatically creates the following files.

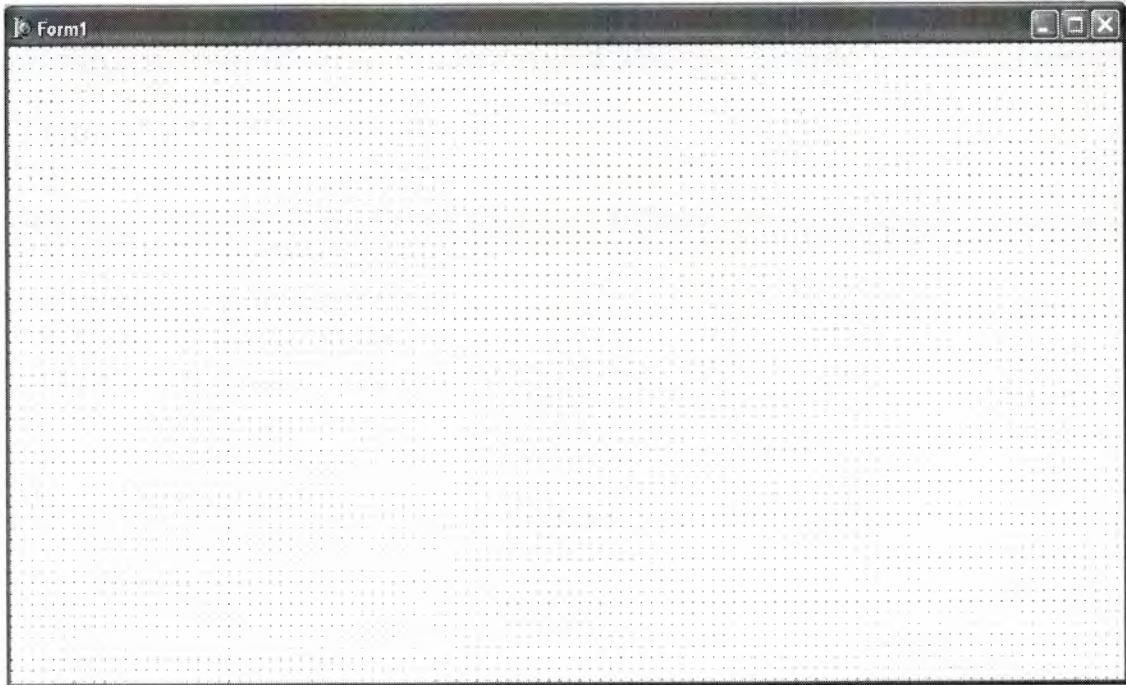
- Project1.DPR : a source-code file associated with the project. This is called a project file.
  - Unit1.PAS : a source-code file associated with the main project form. This is called a unit file.
  - Unit1.DFM : a resource file that stores information about the main project form. This is called a form file.
3. Choose File | Save All to save your files to disk. When the Save dialog appears, navigate to your folder and save each file using its default name.

Later on, you can save your work at any time by choosing File | Save All.



When you save your project, Delphi creates additional files in your project directory. You don't need to worry about them but don't delete them.

When you open a new project, Delphi displays the project's main form, named Form1 by default. You'll create the user interface and other parts of your application by placing components on this form.



**Figure 1.17** Form Screen

The default form has maximize, minimize buttons and a close button, and a control menu

Next to the form, you'll see the Object Inspector, which you can use to set property values for the form and components you place on it.

The drop-down list at the top of the Object Inspector shows the current selected object. When an object is selected the Object Inspector shows its properties.

### **1.5.2 Setting Property Values**

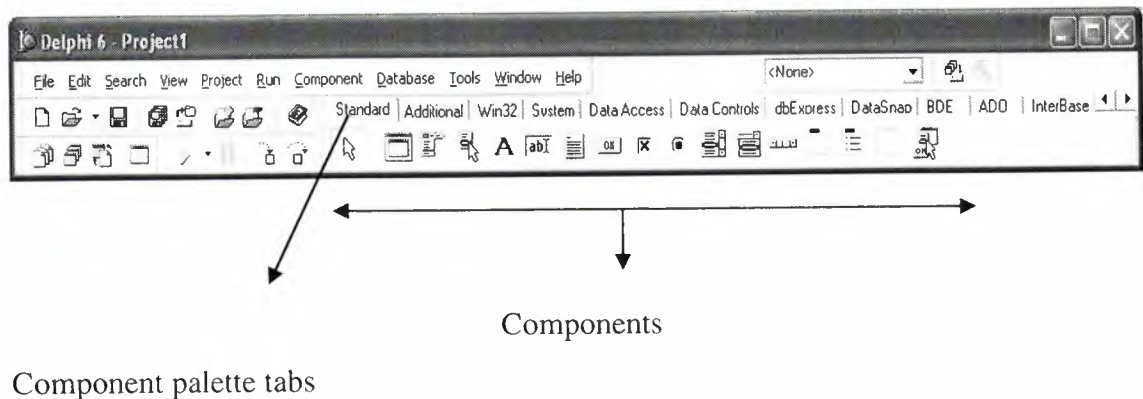
When you use the Object Inspector to set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called design-time settings.

For Example; set the background color of Form1 to Aqua.

Find the form's Color property in the Object Inspector and click the drop-down list displayed to the right of the property. Choose clAqua from the list.

### 1.5.3 Adding objects to the form

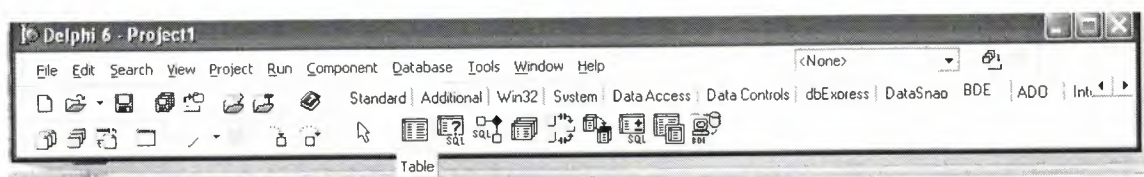
The Component palette represents components by icons grouped onto tabbed pages. Add a component to a form by selecting the component on the palette, then clicking on the form where you want to place it. You can also double-click a component to place it in the middle of the form.



**Figure 1.18** Standard Bar

### 1.5.4 Add a Table and a StatusBar to the Form

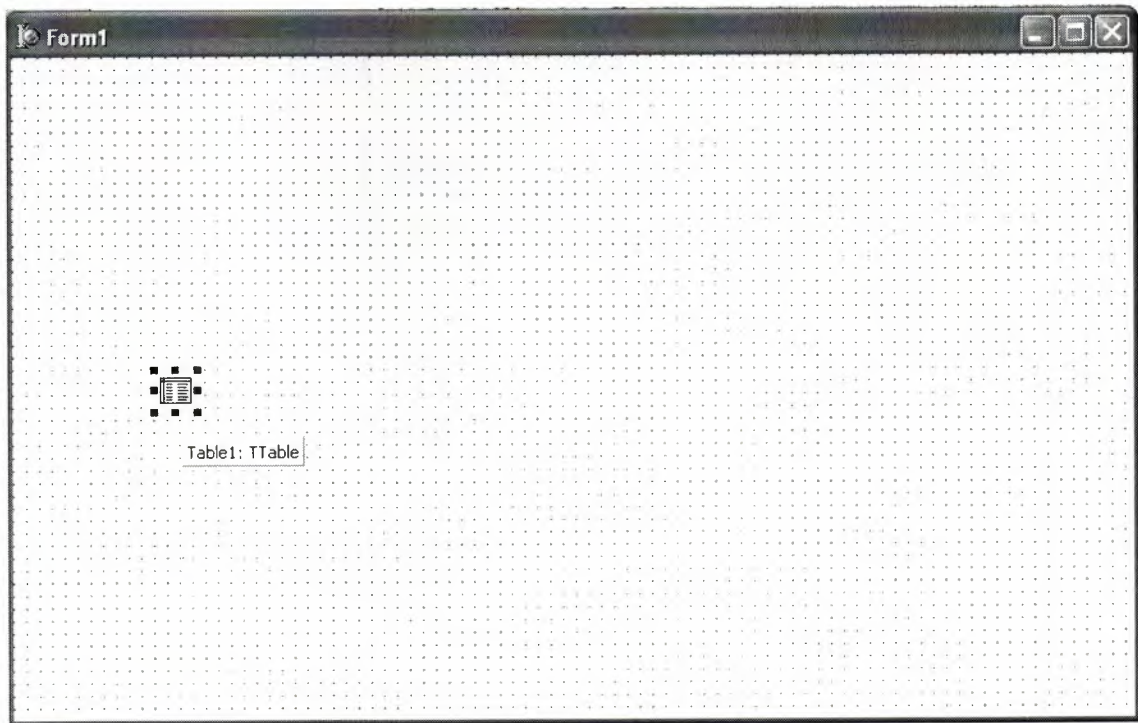
Drop a Table component onto the form. Click the BDE tab on the Component palette. To find the Table component, point at an icon on the palette for a moment; Delphi displays a Help hint showing the name of the component.



**Figure 1.19** BDE Component palette



When you find the Table component, click it once to select it, and then click on the form to place the component. The Table component is non visual, so it doesn't matter where you put it. Delphi names the object Table1 by default. (When you point to the component on the form, Delphi displays its name—Table1—and the type of object it is—Table.)

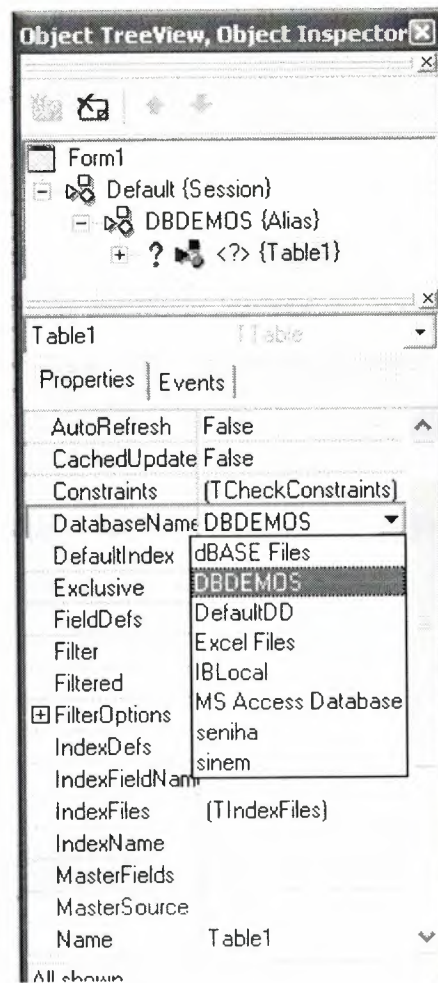


**Figure 1.20** Table in the Form

Each Delphi component is a class; placing a component on a form creates an instance of that class. Once the component is on the form, Delphi generates the code necessary to construct an instance object when your application is running.

Set the DatabaseName property of Table1 to DBDEMOS. (DBDEMOS is an alias to the sample database that you're going to use.)

Select Table1 on the form, and then choose the DatabaseName property in the Object Inspector. Select DBDEMOS from the drop-down list.



**Figure 1.21** Select DatabaseName

Double-click the StatusBar component on the Win32 page of the Component palette. This adds a status bar to the bottom of the application.

Set the AutoHint property of the status bar to True. The easiest way to do this is to double-click on False next to AutoHint in the Object Inspector. (Setting AutoHint to True allows Help hints to appear in the status bar at runtime.)

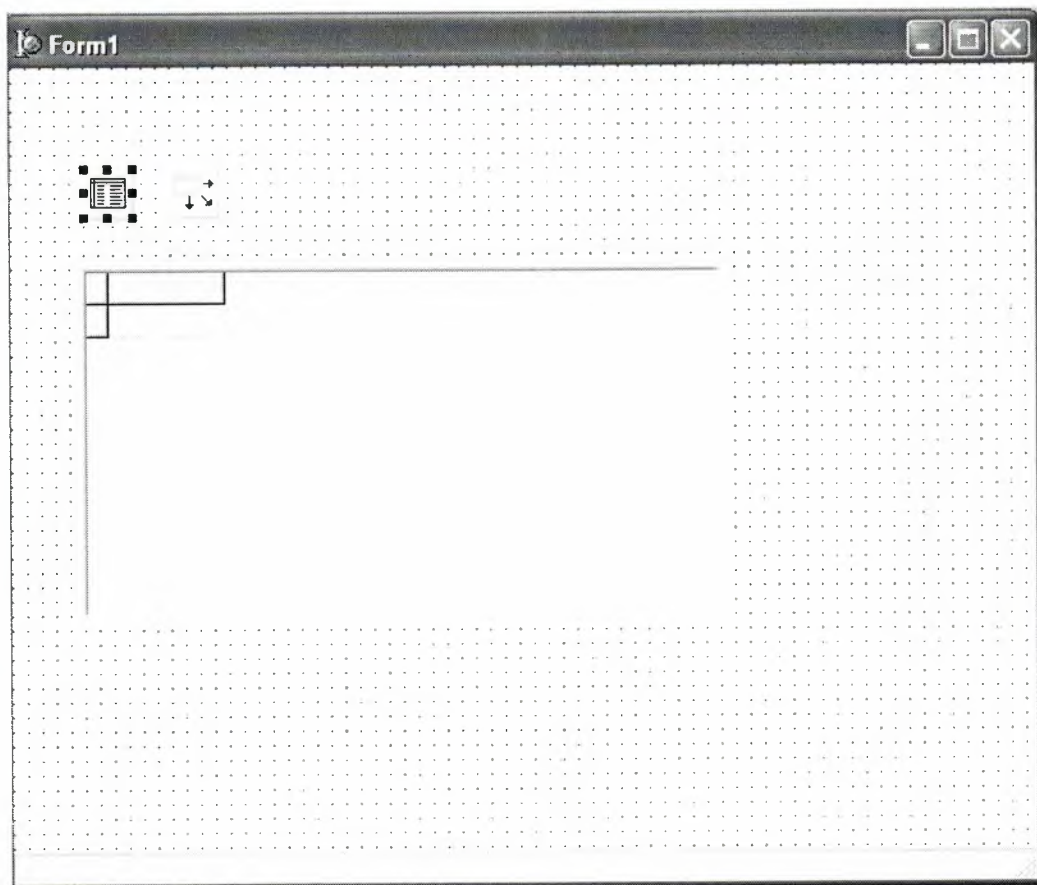
### 1.5.5 Connecting to a Database

The next step is to add database controls and a DataSource to your form.

1. From the Data Access page of the Component palette, drop a DataSource component onto the form. The DataSource component is non visual, so it doesn't matter where you put it on the form. Set its DataSet property to Table1.
2. From the Data Controls page, choose the DBGrid component and drop it onto your form. Position it in the lower left corner of the form above the status bar, and then expand it by dragging its upper right corner.

If necessary, you can enlarge the form by dragging its lower right corner. Your form should now resemble the following figure:

The Data Control page on Component palette holds components that let you view database tables.



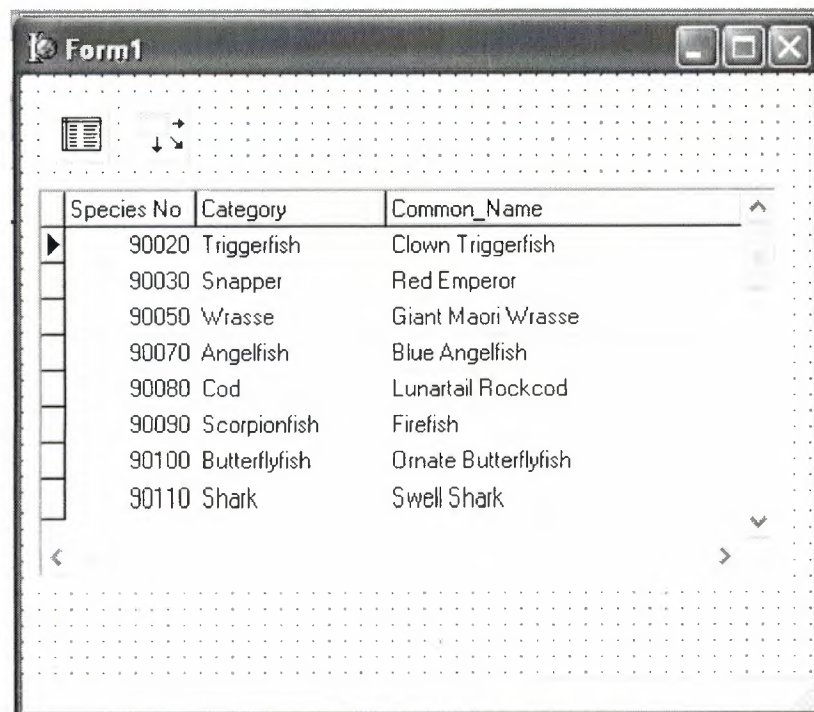
**Figure 1.22** DBGrid in the Form

3. Set DBGrid properties to align the grid with the form. Double-click Anchors in the Object Inspector to display `akLeft`, `akTop`, `akRight`, and `akBottom`; set them all to true.
4. Set the `DataSource` property of DBGrid to `DataSource1` (the default name of the `DataSource` component you just added to the form).

Now you can finish setting up the `Table1` object you placed on the form earlier.

5. Select the `Table1` object on the form, and then set its `TableName` property to `BIOLIFE.DB`. (Name is still `Table1`.) Next, set the `Active` property to `True`.

When you set `Active` to `True`, the grid fills with data from the `BIOLIFE.DB` database table. If the grid doesn't display data, make sure you've correctly set the properties of all the objects on the form, as explained in the instructions above. (Also verify that you copied the sample database files into your `...\Borland Shared\Data` directory when you installed Delphi.)



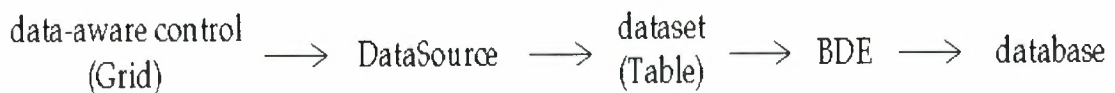
Species No	Category	Common_Name
90020	Triggerfish	Clown Triggerfish
90030	Snapper	Red Emperor
90050	Wrasse	Giant Maori Wrasse
90070	Angelfish	Blue Angelfish
90080	Cod	Lunartail Rockcod
90090	Scorpionfish	Firefish
90100	Butterflyfish	Ornate Butterflyfish
90110	Shark	Swell Shark

**Figure 1.23** Show Table



The DBGrid control displays data at design time, while you are working in the IDE. This allows you to verify that you've connected to the database correctly. You cannot, however, edit the data at design time; to edit the data in the table, you'll have to run the application.

6. Press F9 to compile and run the project. (You can also run the project by clicking the Run button on the Debug toolbar, or by choosing Run from the Run menu.)
7. In connecting our application to a database, we've used three components and several levels of indirection. A data-aware control (in this case, a DBGrid) points to a DataSource object, which in turn points to a dataset object (in this case, a Table). Finally, the dataset (Table1) points to an actual database table (BIOLIFE), which is accessed through the BDE alias DBDEMOS. (BDE aliases are configured through the BDE Administrator.)



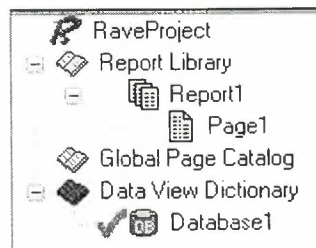
This architecture may seem complicated at first, but in the long run it simplifies development and maintenance. For more information, see “Developing database applications” in the Developer’s Guide or online Help.

## CHAPTER TWO

### 2 THE RAVE REPORTING

#### 2.1 Project Tree

The Project Tree provides an efficient way to visually manage all of the reports in your project. It quickly tells you the structure of your reporting project and the types of components contained on each page with icons that are the same as the component buttons. The Project Tree also visually shows parent-child relationships, the print order of component as well as the current selection (green check marks). You can select components by clicking on the component on the Page in the Visual Designer or on the Project Tree. Non-visual components appear only in the Project Tree in order not to clutter up your report design.



**Figure 2.1** Project Tree

There are three main sections in the Project Tree:

- The Report Library
- The Global Page Catalog
- The Data View Dictionary

Reports themselves can contain any number of page definitions. Global Pages are used to hold items that you want accessible to multiple reports. Data Views contain your field definitions and provide a link to the data in your application.

## 2.2 Design Tools

Rave is all about easy management. Besides making reporting easy and organized, Rave likes to keep itself organized and all according to what you want.

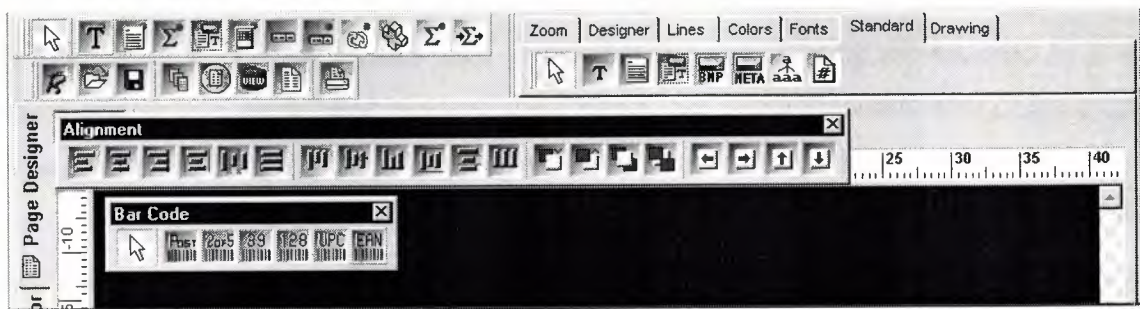


Figure 2.2 Toolbars

Since Rave is designed to be of ease to you there are three easy three ways for you to manage the many toolbars within Rave, which are:

- Tab-docking
- Normal docking
- Free-floating

Rave's many toolbars make it easy to design even the most complicated report. The toolbars include: Project, Designer, Zoom, Alignment, Color, Line, Font, Standard, Drawing, Report and Barcode component toolbars. Since it is possible to create and install new components, you may have other component toolbar buttons in your designer.



Figure 2.3 Project Toolbar

The Project toolbar provides quick access to project level functions such as New Project, Project Open, Project Save, New Report, New Global Page, New Data View, New Report Page or Execute Report.



**Figure 2.4** Designer Toolbar

The Designer toolbar allows you to change the characteristics of the Page in the Visual Designer. Characteristics such as whether the grid is being shown, snap to grid, draw grid on top, show band headers, show rulers, and show the waste area of the page. The last button brings up Rave's extensive Preferences dialog, which is described later.



**Figure 2.5** Zoom Toolbar

When you are working on a report with a complex design, you will find it much easier if you become familiar with the Zoom toolbar, which gives you quick access to Rave's extensive zooming capabilities. Select the zoom percent from a drop down list, type it in or use the Zoom Tool, Zoom In, Zoom Out, Zoom Selected, Zoom Page Width or Zoom Whole Page buttons.



**Figure 2.6** Alignment Toolbar

To help keep your report looking professional, Rave's Alignment toolbar provides access to a whole host of options to micro-manage the components on your page. The Left/Top, Center, Right/Bottom, Center In Parent, Space Equally, Equate Widths/Heights options offer the traditional alignment options. The Move Forward, Move Behind, Bring to Front and Send to Back order movement buttons allow you to change the print order of components and are visually backed up by the listing of the components in the Project Tree. Lastly, the buttons Tap Left, Tap Right, Tap Up and

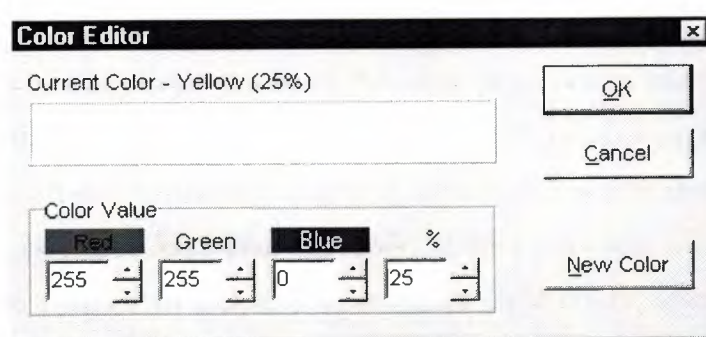


Tap Down allow you to micro-adjust the position of components to the exact position you need.



**Figure 2.7** Colors Toolbar

The Color toolbar allows you to quickly select the primary and secondary colors of your components. There are 8 color spots that you can use to store any custom colors that you will be reusing throughout the project. If the colors available aren't enough, you can double click on the custom color palettes and create a different color using Rave's Color Editor (shown at right). With the Color Editor, you can select from a wider variety of colors or create your own combination of Red, Green and Blue and even select a percent saturation for the current color.



**Figure 2.8** Colors Editor

The Line toolbar is a useful tool for changing the line/border thickness and style for components such as Line and Circle. Sizes are listed in points instead of pixels so that your lines will always be the same thickness on your reports no matter the resolution of the printer that you are using.



**Figure 2.9** Line Toolbar

The Font toolbar provides quick access to a text component's font and alignment properties. It can also be useful for quickly viewing the font options for the currently selected text component(s).



Figure 2.10 Fonts Toolbar

## 2.3 Reuse and Maintenance Tools

Reports often take a large part of the development time for an application. Many times, there are many similarities between the design of separate reports.

This is where Rave's Mirroring technology comes in. When a component is set to mirror another, it assumes the appearance and properties of the component it is mirroring. The two components can be on the same page, across pages within the same report or on a global page. This is the primary purpose of a global page. You can almost think of it like an Object Repository, a central location for you to store reporting items that you want accessible to more than one report. If the component is a container control like TraveSection (similar to Delphi's Tpanel), all child components are mirrored as well. When the original component changes, all mirroring components will also change. While the mirrored component cannot change its properties, you can add additional components if it is a container control.

Here are just a few examples of where Mirroring would be useful:

Your customer wants a standard page header and footer on every page of their 50 reports. Now imagine you have all the reports done and your customer wants to change the layout of the headers and footers.

The Old Way – You would need to open up all 50 report definitions and change them one at a time.

The Rave Way – You would mirror the standard header and footer on each report you create and then any changes would only have to be done in one location. Also, if the standard header included a large bitmap, your reporting project would only contain a single copy rather than the many copies that a traditional report designer would require. You have to replicate a pre-printed form. The problem is there are 6 different variations of this form with only minor differences between each.

The Old Way – Assuming a traditional report designer could even handle this type of report, you would create the first form, cut and paste it into the second, make the minor modifications, then repeat for the other 4 forms, ending up with 6 reports that would be hard to maintain and take up a lot more memory.

The Rave Way – You would first create the common items of the form on a separate page, then mirror those on each form and add the unique parts for each as needed. If anything ever needed to be changed in the common section of the form, you would only need to change it in one place and since you're sharing most of the form's content, the report definitions take up much less room.

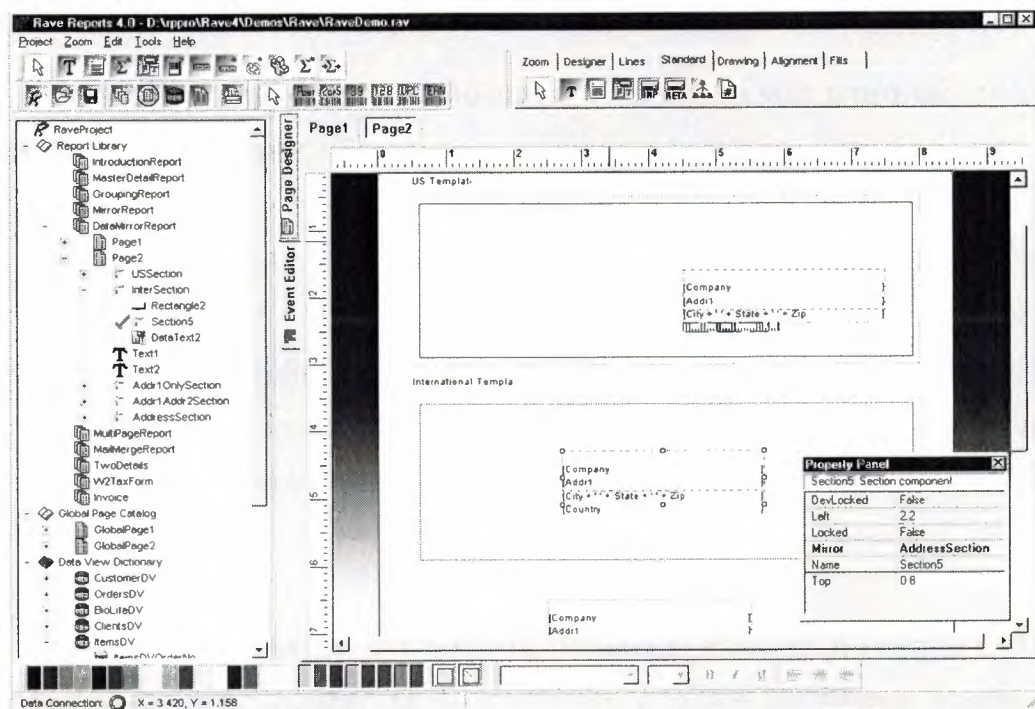


Figure 2.11 Mirror Report Example



Every text component has a FontMirror property which you can assign to a FontMaster component. This will allow you to change the fonts of many text controls from a single location. Imagine having Header, Body and Footer FontMaster components on a global page and changing the appearance of all of your reports with just a few mouse clicks.

Another important aspect of maintaining any large project is documentation. The Project and every Report, Page, Data View and Data Field component has a multi-line Description Property that can be used to comment the intended usage or other information. This can be useful if you are coming back to a project that you last worked on 6 months ago or especially if another programmer or your end user will be modifying reports that you created.

## 2.4 Standard Components



**Figure 2.12** Standard Tool Bar

**Text** – This component is used to display fixed text on your report for items such as column headers or report titles.

**Memo** – This component is used to display fixed text in a word wrapped fashion on your report. Using the MailMergeItems property and the Mail Merge Editor shown below, you can create a mail merge type of report where Rave will replace tokens in the memo text with a replacement string. Note that this replacement string can be edited with the Edit button, which will display the Data Text Editor for quite a bit of extra functionality.

**Section** – This component is a terrific component manager. It acts as a container for other components, in other words it help you to group components together. By



properly using section components and mirroring, you can create reusable and maintainable reports in no time flat.

**Bitmap** – This component is used to display a bitmap (\*.bmp). Through the FileLink property you can reference a file on the hard disk.

**MetaFile** – This component is used to display a metafile (\*.wmf). Through the FileLink property you can reference a file on the hard disk.

**FontMaster** – This component is used to control the font characteristics of any text control through their FontMirror properties. See Reuse and Maintenance for more information.

## 2.5 Drawing Components

**Line** – Draws a diagonal line. (This may not seem like a unique feature but did you know that most Delphi reporting tools cannot create a diagonal line visually.)



**Figure 2.13** Drawing Tool Bar

**Hline** – Draws a horizontal line.

**Vline** – Draws a vertical line.

**Rectangle** – Draws a rectangle.

**Square** – Draws a square.

**Ellipse** – Draws an ellipse.

**Circle** – Draws a circle.

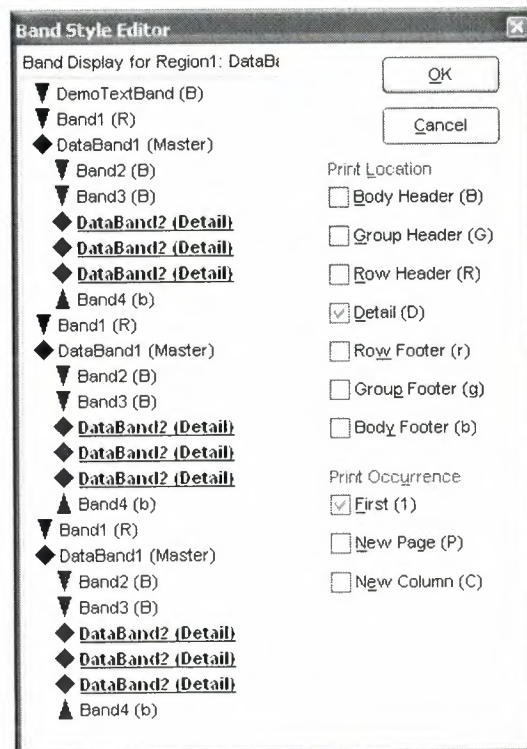
## 2.6 Reporting Components

**Region** – This component acts as a container for Band and DataBand components. To create a composite or sub-report, simply drop more than one region on a page and add the appropriate bands to each.



**Figure 2.14** Report Tool Bar

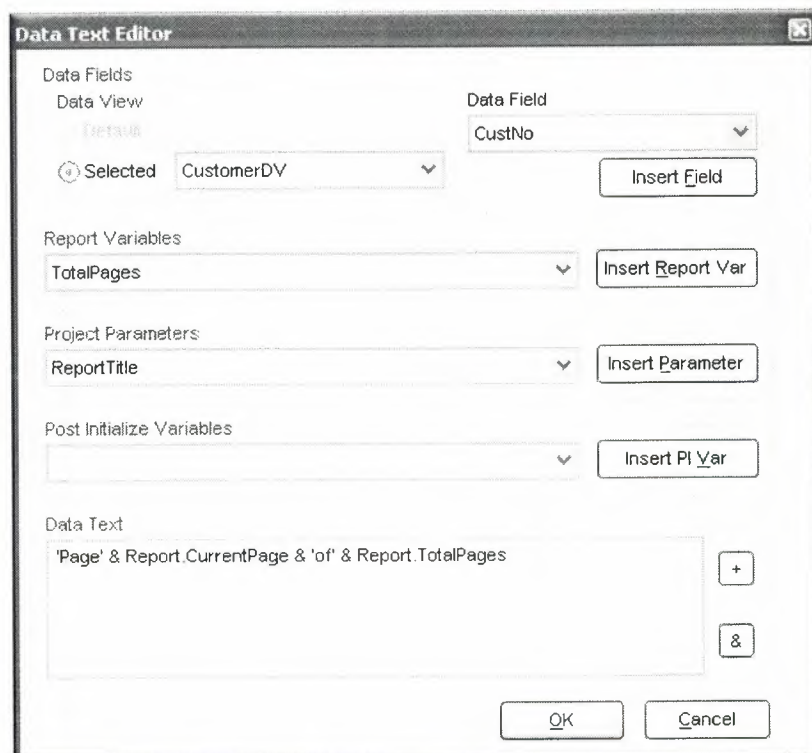
**Band** – This component is primarily used to create header and footer bands in a banded style report. A Band component can only be created within a region and its purpose is controlled through the Band Style Editor shown below. The Band Style Editor displays a virtual layout of all of your bands for the given print locations of each band or data band. Note that you can create as many Bands as you like and a Band may print in multiple locations if the report design requires it. So for example, if you want a solid horizontal line to appear above and below a detail body, you could create a single band and set it to print on both the Body Header and Body Footer. You can also control the Print Occurrence for a Band, having it continue on a new page or column or any combination of occurrence settings. You can set a Band to group on specific fields and can create as many different types of group headers or footers as your report requires. Basically, with Rave's Band and DataBand components, you'll be able to create just about any banded style layout that you can imagine.



**Figure 2.15** Band Style Editor

**DataBand** – The DataBand component is fairly similar to a band component except that it is tied to a particular DataView and iterates across the rows in the DataView. You can link DataBands together for Master-Detail to unlimited levels or multiple details on the same level. Some advanced features that are supported by a DataBand include KeepBodyTogether, KeepRowTogether, StartNewPage, MaxRows and Orphan/Widow control.

**DataText** – The DataText component is the primary means to output fields from your database. You can quickly select a specific DataView and DataField with Property Panel or use the Data Text Editor shown below to create any combination of string constants, data fields, report variables or project parameters. The & concatenation operator is the same as the + operator, except that it also inserts a space. Report Variables are items such as total pages or current date and time in a variety of formats. Project Parameters are custom variables that you create and initialize from your Delphi application. Project Parameters can be used for items such as user defined report titles, printing the current user name or other custom information.

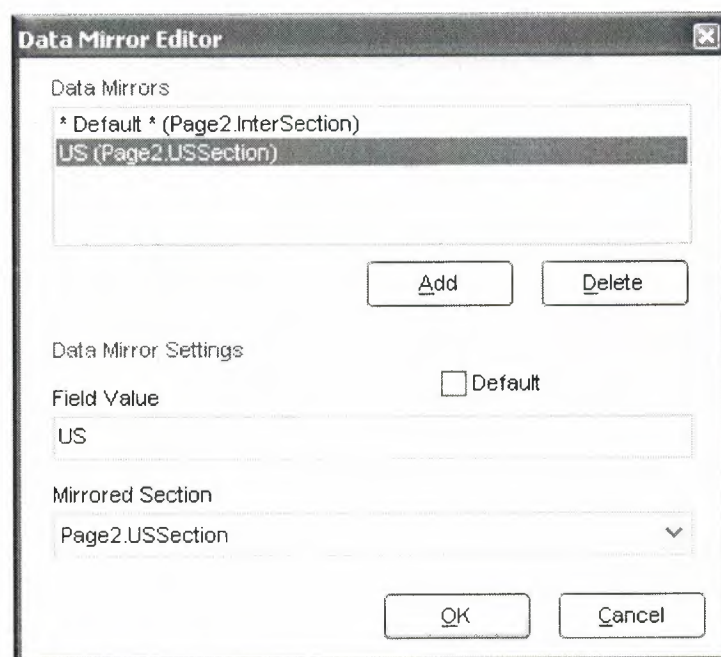


**Figure 2.16** Data Text Editor

**DataMemo** – This component is very similar to the Memo component except that it retrieves data from a DataField. DataMemo component's print text data out in a word wrapped fashion and the DataField can be any text type, not just memo fields. It also has RTF and mail merge support.

**CalcText** – This component is used to perform simple operations such as Sum, Average, Count, Min and Max on a data field. You can set the value as a running total and place it in any type of band or anywhere on the page) you need it.

**DataMirrorSection** – The data mirror section component is similar to Rave's section component (found in the Standard Toolbar) with one major difference, it will dynamically mirror another section depending upon the value of a DataField. You configure the data mirror section using the Data Mirror Editor (shown below). This component is very useful for printing out data that has different formats depending upon the type of data. One example is an address field that could print a US format if the country field is "US" and an international format otherwise (using the Default option in the Data Mirror Editor). You could also print Boolean field values with your own custom bitmaps.



**Figure 2.17** Data Mirror Editor



## 2.7 Barcode Components



Figure 2.18 Barcode Toolbar

PostNetBarCode – Prints a US PostNet bar code.

I2of5BarCode – Prints Interleaved 2 of 5 barcodes.

Code39BarCode – Prints standard and extended Code 39 barcodes.

Code128BarCode – Prints A, B and C Code 128 barcodes.

UPCBarCode – Prints UPC-12 barcodes.

EANBarCode – Prints EAN-13 barcodes.

## 2.8 Anchors

Anchors are a powerful way to create a report that dynamically adjusts to changing sizes. This allows you to create reports that can print well whether the user selects landscape or portrait, 8.5" by 11" or A4. There are 6 different anchor values for both the horizontal and vertical dimensions to allow you to control each component in exactly the manner that it needs. The Anchor Editor (shown at right) even shows you a helpful bitmap of how each anchor setting works.

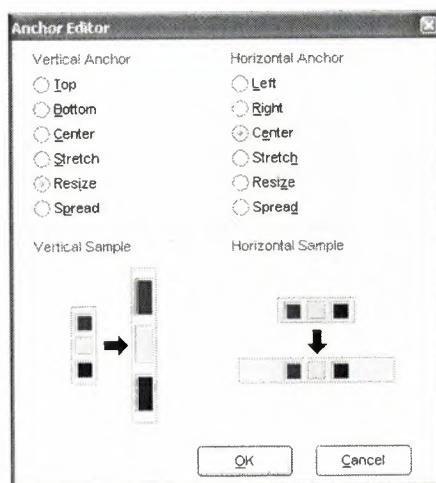


Figure 2.19 Anchor Editor

## 2.9 Code Based Reports

Lately Delphi has decided to include Rave Reports as the default reporting solution, replacing Quick Reports. Since they work in very different paradigms, many people were confused by the new environment. This is intended as an introduction for people who haven't worked with Rave yet, and would like to start.

Nowadays Delphi ships with Rave Reports 5.0.8. If you haven't already, download the update from the registered users page, since it fixes some important problems.

You can develop reports with Rave using two different ways: Code Based or with the Visual Designer.

With Code Based, you write reports using plain Delphi code. That provides a very flexible way displaying any kind of data, allowing any kind of complex layouts.

To write a code based report, just drop a TrvSystem component on the form and write the report on the OnPrint event handler. Sender is the report you are creating, and can be typecasted to TbaseReport. It contains all the methods you need to output information to that particular report.

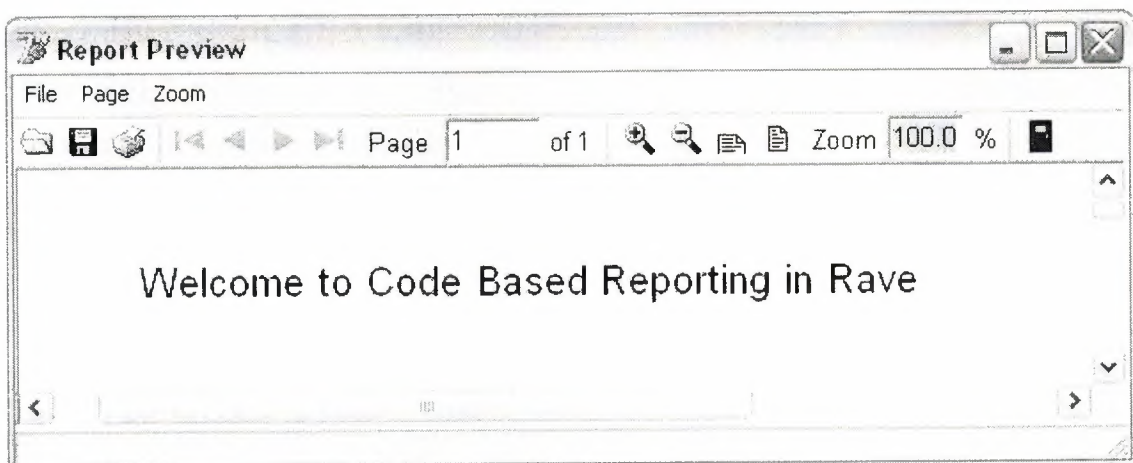
### 2.9.1 Simple Code Base Report

Here's a simple report using the code based mechanism:

```
procedure TFormMain.RvSystemPrint(Sender: Tobject);
begin
  with Sender as TbaseReport do
  begin
    SetFont('Arial', 15);
    GotoXY(1,1);
    Print('Welcome to Code Based Reporting in Rave');
  end;
end;
```

To execute this report, call `RvSystem.Execute` method.

So, what does that simple code do? First, it calls `SetFont` to select the font and size of the text that will be printed from that point on. Then it positions the cursor on the coordinates (1,1). These coordinates are expressed using the units set in the `SystemPrinter.Units` property of the `RvSystem` object, and it defaults to Inches. You can set it to `unUser` and set a number relative to Inches in the `SystemPrinter.UnitsFactor` property. For example, if `UnitsFactor` was set to 0.5 then 1 unit would correspond to half an inch. Finally, the code calls the `Print` method to output the text. Here's the output:



**Figure 2.20** Report Preview

### 2.9.2 Tabular Code Based Report

Here's another example. It displays a list of the folders in the root of the current drive, along with a recursive count of number of files and folder, and total size of the files included in each folder.

```
Procedure TFormMain.PrintTabularReport(Report: TbaseReport);  
var  
    FolderList : TstringList;  
    I          : Integer;  
    NumFiles   : Cardinal;
```

```

NumFolders : Cardinal;
SizeFiles : Cardinal;
Root      : string;
begin
  with Report do
  begin
    SetFont('Arial', 15);
    NewLine;
    PrintCenter('List of Folders in the Drive Root', 4);
    NewLine;
    NewLine;
    ClearTabs;
    SetTab(0.2, pjLeft, 1.7, 0, 0, 0);
    SetTab(1.7, pjRight, 3.1, 0, 0, 0);
    SetTab(3.1, pjRight, 3.5, 0, 0, 0);
    SetTab(3.5, pjRight, 4.5, 0, 0, 0);
    SetFont('Arial', 10);
    Bold := True;
    PrintTab('Folder Name');
    PrintTab('Number of Files');
    PrintTab('Number of Folders');
    PrintTab('Size of Files');
    Bold := False;
    NewLine;
    FolderList := TStringList.Create;
  try
    Root := IncludeTrailingPathDelimiter(ExtractFileDrive(ParamStr(0)));
    EnumFolders(FolderList, Root);
    for I := 0 to FolderList.Count - 1 do
    begin
      PrintTab(FolderList[I]);
      GetFolderInfo(IncludeTrailingPathDelimiter(Root+FolderList[I]),
        NumFiles, NumFolders, SizeFiles);
    end
  end
end

```

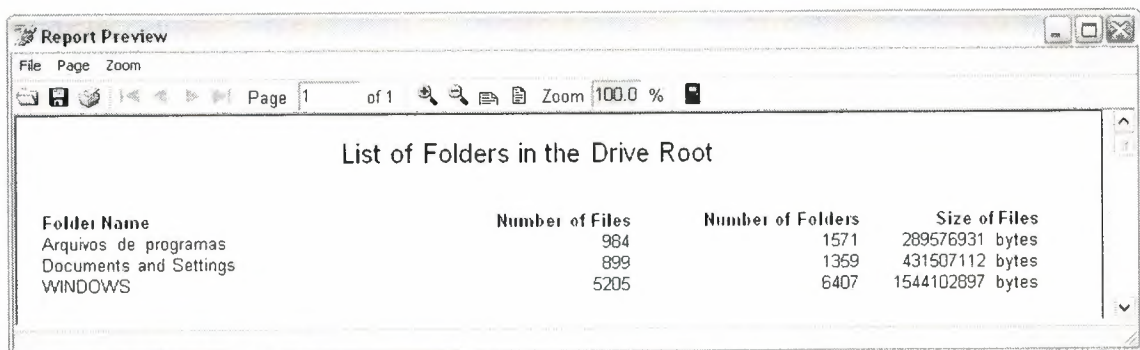


```

PrintTab(Format('%u',[NumFiles]));
PrintTab(Format('%u',[NumFolders]));
PrintTab(Format('%u bytes',[SizeFiles]));
NewLine;
end;
finally
  FolderList.Free;
end;
end;
end;
end;

```

Notice that a different approach has been taken: instead of specifying the coordinates of each text output, the printing was done using Lines and Columns as references. The line height depends on the size of the current font: each unit represents 1/72nds of an inch, so each line printed with a size 10 font will have, appropriately, a height of 0.138 inches. Lines are advanced after calls to PrintLn or NewLine. Columns are defined using calls to the SetTabs method, and the PrintTab method will print the text in the current column and advance to the next one. Here's the output:



The screenshot shows a window titled 'Report Preview' with a menu bar (File, Page, Zoom) and a toolbar. The main content area displays a table titled 'List of Folders in the Drive Root'. The table has four columns: 'Folder Name', 'Number of Files', 'Number of Folders', and 'Size of Files'. The data rows are: 'Arquivos de programas' (984 files, 1571 folders, 289576931 bytes), 'Documents and Settings' (899 files, 1359 folders, 431507112 bytes), and 'WINDOWS' (5205 files, 6407 folders, 1544102897 bytes).

Folder Name	Number of Files	Number of Folders	Size of Files
Arquivos de programas	984	1571	289576931 bytes
Documents and Settings	899	1359	431507112 bytes
WINDOWS	5205	6407	1544102897 bytes

**Figure 2.21** Report Preview

### 2.9.3 Graphical Code Based Report

You can include shapes and images in your code based report, along with the text. The following example demonstrates that:

```
procedure TFormMain.PrintGraphicsReport(Report: TbaseReport);
```

```
var
```

```

    Bitmap : Tbitmap;
begin
    with Report do
    begin
        Canvas.Brush.Color := clGray;
        Rectangle(0.3, 0.3, 4.7, 3.3);
        SetFont('Arial', 15);
        FontColor := clRed;
        PrintXY(0.5,0.5, 'Just look at all the graphics!');
        Bitmap := Tbitmap.Create;
    try
        Bitmap.LoadFromFile('delphi.bmp');
        PrintBitmap(3.5,0.3,1,1, Bitmap);
        PrintBitmap(1,2,3,3, Bitmap);
        Canvas.Pen.Color := clBlue;
        Canvas.Brush.Bitmap := Bitmap;
        Ellipse(5,0.3,6,3.3);
        Ellipse(2,1,4,1.9);
    finally
        Bitmap.Free;
    end;
    Canvas.Pen.Color := clBlack;
    Canvas.Brush.Style := bsSolid;
    Canvas.Brush.Color := clYellow;
    Pie(0.7,0.7,1.7,1.7,1,1,1,2);
    Canvas.Brush.Color := clGreen;
    Pie(0.7,0.7,1.7,1.7,1,2,1,1);
    end;
end;

```

In this example the methods Rectangle, Ellipse and Pie have been used draw shapes with different fills. Bitmaps were outputted using PrintBitmap and as the brush of the ellipses. Here's the output:

## Graphics Report Example

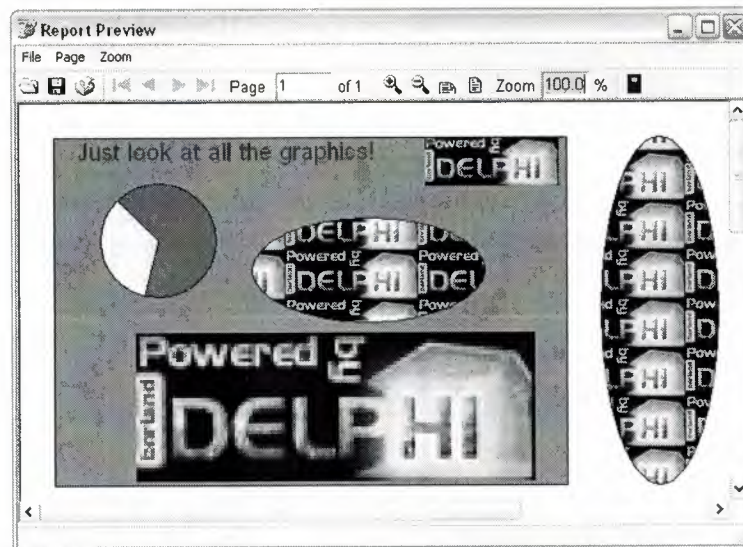


Figure 2.22 Report Preview

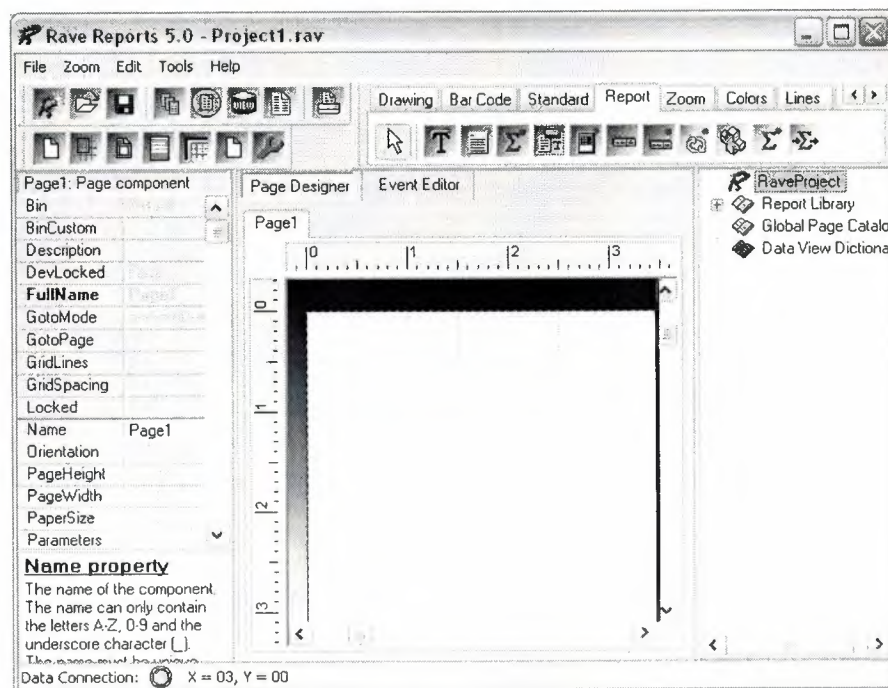
## 2.10 Visually Designed Reports

### 2.10.1 The Visual Designer

If you are used to work with Quick Reports, the default reporting engine included in the previous versions of Delphi, you created your reports using Delphi's own form designer, and they were save in the DFM, included as resources in your executable. Rave works a bit differently in this aspect: it has it's own report designer, and saves the report using it's own file format. This has some advantages, including the fact that your reports can be made "standalone", and be used or updated independently of your application, or even made available in a Intranet or in the Internet, using Nevrona's Rave Report Server. Of course, you can still have it saved in a form's DFM.

To get started with the Rave Visual Designer, drop a TrvProject in a form. This will be the link from your application to the reports you are developing. If you want, you can add a TrvSystem and link your RvProject to it, through it's Engine property. The RvSystem is the object responsible for the general configuration of the reports: the printer that is going to be used, the margins, the number of pages, and so on. To start a new project, double click the RvProject you added to the form, or select "Rave Visual Designer" from its context menu.

This is the interface that you will be working on:



**Figure 2.23** Rave Visual Designer

The interface is simple, and you might be familiar with some parts of it from Delphi's IDE. On the top there's the menu, the toolbar, and the component palette that contain the components that will be used in the reports. In the left there's the Object Inspector, which will be used to adjust the properties of the components of the report. In the middle there's the Page Designer or the Event Editor, and in the left there's the very usefull Project Treeview. For a quick overview of the components in the palette, you can go to Nevrona's Visual Designer page.

A Rave Project File can have one or more reports. That way you can keep common items between them in a single location, called Global Pages. If you expand the Report Library node of the Project Treeview, you can see that right now you are working on Report1. Clicking on it, its properties will show on the Inspector. Let's change it's name and call it SimpleReport. Next, go to the Standard tab on the Component Palette, and pick a Text component and add it to the page. Change its text property, and adjust its size and position. Here's how it looks like:





**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**Stock Property by Using Delphi**

**Graduation Project**

**COM 400**

**Student: Seda ONHAN (20032905)**

**Supervisor: Assist. Prof. Dr. Imanov ELBRUS**

**Nicosia – 2008**





## ACKNOWLEDGEMENT

*First of all, I would like give my special thanks to my supervisor Assist. Prof. Dr. Imanov ELBRUS. He helped and supported me to complete my project by any means of necessary. In addition to this he never doubted about me, he always believed in me that I will fulfill and succeed on my project. I am glad to that I did not disappoint him.*

*Furthermore, I want to give my special thanks and best regards to my parents. They were always kind and patient to me. I wouldn't be here without their endless support.*

*Finally, I want to give my special thanks to my friends whose are Cemal Kavalcioğlu, Selman Oğuzhan ESER. They are supported and helped me to complete my project. I am very happy to have such friends.*

## **ABSTRACT**

The aim of this Project is to record the stock device for any Properties Company. The program was prepared by using Delphi 7 programming and using Paradox7. Delphi is a programming language that can be used with Paradox7.

This project consists of many different pages but most of them depended each other Initially, SIGN IN form comes to screen. Afterwards the Main menu of Properties Company comes to screen. After Main Menu you are going to see the main form that contains 15 main menus.

## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENT .....</b>	<b>I</b>
<b>ABSTRACT.....</b>	<b>II</b>
<b>TABLE OF CONTENTS .....</b>	<b>III</b>
<b>INTRODUCTION .....</b>	<b>1</b>

### CHAPTER ONE : BASIC CONCEPT OF DELPHI

1.1 Introduction to Delphi.....	2
1.2 What is Delphi? .....	2
1.2.1 Delphi Compilers .....	2
1.2.2 What kind of programming can you do with Delphi? .....	3
1.2.3 History of Delphi .....	4
1.2.4 Advantages & Disadvantages Delphi .....	6
1.3 Delphi 6 Editions .....	7
1.3.1 Delphi 6 Architect.....	7
1.3.2 Installation Delphi 6.....	8
1.4 A Tour of the Environment.....	10
1.4.1 Running Delphi for the First Time .....	10
1.4.2 The Delphi IDE.....	11
1.4.3 The Menus & Toolbar.....	12
1.4.4 The Component Palette.....	12
1.4.5 The Code Editor.....	13
1.4.6 The Object Inspector.....	14
1.4.7 The Object TreeView.....	15
1.4.8 Class Completion.....	16
1.4.9 Debugging applications .....	17
1.4.10 Exploring Databases .....	18
1.4.11 Templates and the Object Repository .....	19
1.5 Programming with Delphi .....	20
1.5.1 Starting a New Application.....	20
1.5.2 Setting Property Values .....	21
1.5.3 Adding objects to the form .....	22
1.5.4 Add a Table and a StatusBar to the Form.....	22
1.5.5 Connecting to a Database .....	24

### CHAPTER TWO : THE RAVE REPORTING

2.1 Project Tree.....	28
2.2 Design Tools .....	29
2.3 Reuse and Maintenance Tools .....	32
2.4 Standard Components .....	34
2.5 Drawing Components .....	35
2.6 Reporting Components .....	35
2.7 Barcode Components .....	39
2.8 Anchors .....	39



2.9	Code Based Reports .....	40
2.9.1	Simple Code Base Report .....	40
2.9.2	Tabular Code Based Report .....	41
2.9.3	Graphical Code Based Report.....	43
2.10	Visually Designed Reports .....	45
2.10.1	The Visual Designer .....	45
2.10.2	Interacting with the Project.....	48
2.11	Data Aware Reports.....	55
2.11.1	The Database Connection .....	55
2.11.2	The Driver Data View.....	55
2.11.3	Regions and Bands.....	58
2.11.4	Adding Fields.....	60
2.11.5	Adding the Report to Your Project .....	60

## CHAPTER THREE : STOCK PROPERTY BY USING DELPHI

3.1	Database Connection Screen .....	61
3.2	Main Menu.....	63
3.3	House to Let Menu .....	64
3.3.1	House to Let Organize Form .....	64
3.3.2	House to Let Search Form .....	65
3.3.3	House to Let Report Form .....	68
3.4	House for Sale Menu .....	69
3.4.1	House for Sale Organize Form .....	69
3.4.2	House for Sale Search Form .....	70
3.4.3	Hose for Sale Report Form .....	73
3.5	Shop to Let Menu .....	74
3.5.1	Shop to Let Organize Form .....	74
3.5.2	Shop to Let Search Form .....	75
3.5.3	Shop to Let Report Form .....	78
3.6	Shop for Sale Menu .....	79
3.6.1	Shop for Sale Organize Form .....	79
3.6.2	Shop for Sale Search Form .....	80
3.6.3	Shop for Sale Report Form .....	83
3.7	Plot to Let Menu .....	84
3.7.1	Plot to Let Organize Form .....	84
3.7.2	Plot to Let Search Form .....	85
3.7.3	Plot to Let Report Form .....	88
3.8	Garden for Sale Menu.....	89
3.8.1	Garden for Sale Organize Form.....	89
3.8.2	Garden for Sale Search Form.....	90
3.8.3	Garden for Sale Report Form.....	93
3.9	Building for Sale Menu.....	94
3.9.1	Building for Sale Organize Form.....	94
3.9.2	Buildig for Sale Search Form .....	95
3.9.3	Building for Sale Report Form .....	98
3.10	Farm for Sale Menu .....	99
3.10.1	Farm for Sale Organize Form .....	99

3.10.2	Farm for Sale Search Form .....	100
3.10.3	Farm for Sale Report Form .....	103
3.11	Villa for Sale Menu .....	104
3.11.1	Villa for Sale Organize Form .....	104
3.11.2	Villa for Sale Search Form .....	105
3.11.3	Villa for Sale Report Form .....	108
3.12	Field for Sale Menu .....	109
3.12.1	Field for Sale Organize Form .....	109
3.12.2	Filed for Sale Search Form .....	110
3.12.3	Field for Sale Report Form .....	113
3.13	Flier Print Menu .....	114
3.13.1	Flier Print Organize Form .....	114
3.13.2	House to Let Advertisements Form .....	115
3.13.3	Villa for Sale Advertisements Form .....	116
3.13.4	Shop to Let Advertisements Form .....	117
3.13.5	Plot for Sale Advertisements Form .....	118
3.13.6	House for Sale Advertisements Form .....	119
3.13.7	Field for Sale Advertisements Form .....	120
3.13.8	Shop for Sale Advertisements Form .....	121
3.13.9	Garden for Sale Advertisements Form .....	122
3.13.10	Building for Sale Advertisements Form .....	123
3.13.11	Farm for Sale Advertisements Form .....	124
3.14	User Register Menu .....	125
3.15	About Menu .....	126
3.16	Informations Menu .....	127
3.17	Exit Menu .....	128
<b>CONCLUSION</b> .....		129
<b>REFERENCES</b> .....		130
<b>APPENDIX</b> .....		140

## INTRODUCTION

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

The project consists of the introduction, three chapters, and conclusion.

- Chapter one describes Basic Concept of Delphi.
- Chapter two describes the database that uses Delphi programming language.
- Chapter three explains Stock Property by Using Delphi.

## **CHAPTER ONE**

### **1 BASIC CONCEPT OF DELPHI**

#### **1.1 Introduction to Delphi**

Although I am not the most experienced or knowledgeable person on the forums I thought it was time to write a good introductory article for Delphi

#### **1.2 What is Delphi?**

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

For the purposes of this series I will be using Delphi 6. Delphi 6 provides all the tools you need to develop test and deploy Windows applications, including a large number of so-called reusable components.

Borland Delphi provides a cross platform solution when used with Borland Kylix – Borland's RAD tool for the Linux platform.

##### **1.2.1 Delphi Compilers**

There are two types compiler for Delphi

- Turbo Delphi: Free industrial strength Delphi RAD (Rapid Application Development) environment and compiler for Windows. It comes with 200+ components and its own Visual Component Framework.



- Turbo Delphi for .NET: Free industrial strength Delphi application development environment and compiler for the Microsoft .NET platform.

### **1.2.2 What kind of programming can you do with Delphi?**

The simple answer is “more or less anything”. Because the code is compiled, it runs quickly, and is therefore suitable for writing more or less any program that you would consider a candidate for the Windows operating system.

You probably won't be using it to write embedded systems for washing machines, toasters or fuel injection systems, but for more or less anything else, it can be used (and the chances are that probably someone somewhere has!)

Some projects to which Delphi is suited:

- Simple, single user database applications
- Intermediate multi-user database applications
- Large scale multi-tier, multi-user database applications
- Internet applications
- Graphics Applications
- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools
- Communications tools using the Internet, Telephone or LAN
- Web based applications

This is not intended to be an exhaustive list, more an indication of the depth and breadth of Delphi's applicability. Because it is possible to access any and all of the Windows API, and because if all else fails, Delphi will allow you to drop a few lines of assembler code directly into your ordinary Pascal instructions, it is possible to do more or less anything. Delphi can also be used to write Dynamically Linked Libraries (DLLs) and can call out to DLLs written in other programming languages without difficulty.

Because Delphi is based on the concept of self contained Components (elements of code that can be dropped directly on to a form in your application, and exist in object form, performing their function until they are no longer required), it is possible to build applications very rapidly. Because Delphi has been available for quite some time, the number of pre-written components has been increasing to the point that now there is a component to do more or less anything you can imagine. The job of the programmer has become one of gluing together appropriate components with code that operates them as required.

### **1.2.3 History of Delphi**

Delphi was one of the first of what came to be known as "RAD" tools, for Rapid Application Development, when released in 1995 for the 16-bit Windows 3.1. Delphi 2, released a year later, supported 32-bit Windows environments, and a C++ variant, C++ Builder, followed a few years after.

The chief architect behind Delphi, and its predecessor Turbo Pascal, was Anders Hejlsberg until he was headhunted in 1996 by Microsoft, where he worked on Visual J++ and subsequently became the chief designer of C Sharp programming language C# and a key participant in the creation of the Microsoft .NET Framework.

In 2001 a Linux version known as Kylix programming tool Kylix became available. However, due to low quality and subsequent lack of interest, Kylix was abandoned after version 3.

Support for Linux and Windows cross platform development (through Kylix and the CLX component library) was added in 2002 with the release of Delphi 6.

Delphi 8, released December 2003, was a .NET -only release that allowed developers to compile Delphi Object Pascal code into .NET Microsoft Intermediate Language MSIL. It was also significant in that it changed its IDE for the first time, from the multiple-floating-window-on-desktop style IDE to a look and feel similar to Microsoft's Visual Studio.NET.

Although Borland fulfilled one of the biggest requests from developers (.NET support), it was criticized both for making it available too late, when a lot of former Delphi developers had already moved to C#, and for focusing so much on backward compatibility that it was not very easy to write new code in Delphi. Delphi 8 also lacked significant high-level features of the c sharp, C# language, as well as many of the more appealing features of Microsoft's Visual Studio IDE. (There were also concerns about the future of Delphi Win32 development. Because Delphi 8 did not support Win32, Delphi 7.1 was included in the Delphi 8 package.)

The next version, Delphi 2005 (Delphi 9), included the Win32 and .NET development in a single IDE, reiterating Borland's commitment to Win32 developers. Delphi 2005 includes design-time manipulation of live data from a database. It also includes an improved IDE and added a "for ... in" statement (like C#'s for each) to the language. However, it was criticized by some for its bugs; both Delphi 8 and Delphi 2005 had stability problems when shipped, which were only partially resolved in service packs.

In late 2005, Delphi 2006 was released and federated development of C# and Delphi.NET, Delphi Win32 and C++ into a single IDE. It was much more stable than Delphi 8 or Delphi 2005 when shipped, and improved even more after the service packs and several hot fixes.

On February 8, 2006, Borland announced that it was looking for a buyer for its IDE and database line of products, which include Delphi, to concentrate on its Application Lifecycle Management ALM line. The news met with voluble optimism from the remaining Delphi users.

On September 6, 2006, The Developer Tools Group (the working name of the not yet spun off company) of Borland Software Corporation released single language versions of Borland Developer Studio, bringing back the popular "Turbo" moniker. The Turbo product set includes Turbo Delphi for Win32, Turbo Delphi for .NET, Turbo C++, and Turbo C#. Each version is available in two editions: "Explorer" a free downloadable version and "Professional" a relatively cheap (US\$399) version which

opens access to thousands of third-party components. Unlike earlier “Personal” editions of Delphi, new “Explorer” editions can be used for commercial development.

On November 14, 2006, Borland announced the cancellation of the sale of its Development tools; instead of that it would spin them off into an independent company named “CodeGear”

#### **1.2.4 Advantages & Disadvantages Delphi**

Delphi exhibits the following advantages:

- Rapid Application Development (RAD)
- Based on a well-designed language – high-level and strongly typed, with low-level escapes for experts
- A large community on Usenet and the World Wide Web (e.g. [news://newsgroups.borland.com](mailto:news://newsgroups.borland.com) and Borland’s web access to Delphi)
- Can compile to a single executable, simplifying distribution and reducing DLL versioning issues
- Many VCL and third-party components (usually available with full source code) and tools (documentation, debug tools, etc.)
- Quick optimizing compiler and ability to use assembler code
- Multiple platform native code from the same source code
- High level of source compatibility between versions
- Cross Kylix – a third-party toolkit which allows you to compile native Kylix/Linux applications from inside the Windows Delphi IDE, hence easily enabling dual-platform development and deployment
- Cross FBC – a sister project to Cross Kylix, which enables you to cross-compile your Windows Delphi applications to multi-platform targets – supported by the Free Pascal compiler – without ever leaving the Delphi IDE
- Class helpers to bridge functionality available natively in the Delphi RTL, but not available in a new platform supported by Delphi
- The language’s object orientation features only class- and interface-based Polymorphism in object-oriented programming polymorphism



Disadvantages:

- Limited cross-platform capability for Delphi itself. Compatibles provide more architecture/OS combinations
- Access to platform and third party libraries require header files to be translated to Pascal. This creates delays and introduces the possibilities of errors in translation.
- There are fewer published books on Delphi than on other popular programming languages such as C++ and C#
- A reluctance to break any code has lead to some convoluted language design choices, and orthogonally and predictability have suffered

### **1.3 Delphi 6 Editions**

There are 3 editions in Delphi 6:

- Delphi Personal – makes learning to develop non-commercial Windows applications fast and fun. Delphi 6 Personal makes learning Windows development easy with drag-and-drop visual programming.
- Delphi Professional – adds the tools necessary to create applications with the latest Windows® ME/2000 look-and-feel. Dramatically enhance functionality with minimal code using the power and flexibility of SOAP and XML to easily integrate Web Services into client-side applications.
- Delphi Enterprise – includes additional tools, extensive options for Internet. Delphi 6 makes next-generation e-business development with Web Services a snap.

This Program will concentrate on the Enterprise edition.

#### **1.3.1 Delphi 6 Architect**

Delphi 6 Architect is designed for professional enterprise developers who need to adapt quickly to changing business rules and manage sophisticated applications that synchronize with multiple database schemas. Delphi 2006 Architect includes an advanced ECO III framework that allows developers to rapidly deploy scalable external facing Web applications with executable state diagrams, object-relational mapping, and transparent persistence.

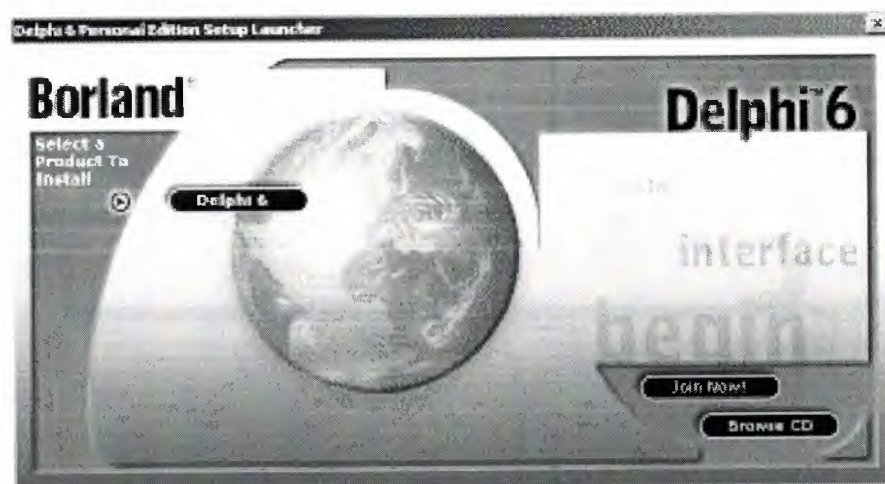
Delphi 6 Architect includes all of the capabilities of the Enterprise edition, and includes the complete ECO III framework, including new support for ECO State Machines powered by State Chart visual diagrams, and simultaneous persistence to multiple and mixed database servers.

- State Chart Diagrams
- Executable ECO State Machines
- Multi- and Mixed- ECO database support

### 1.3.2 Installation Delphi 6

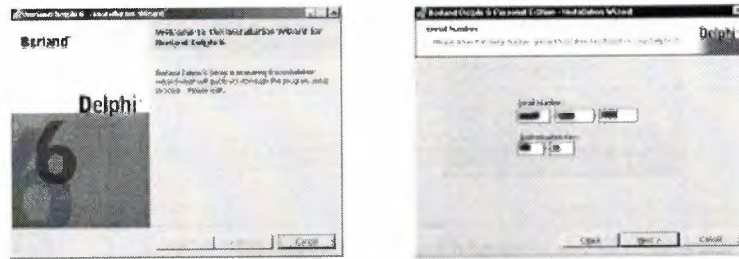
To install Delphi 6 Enterprise, run INSTALL.EXE (default location C:\Program Files\Borland Delphi) and follow the installation instructions.

We are prompted to select a product to install; you only have one choice “Delphi 6”:



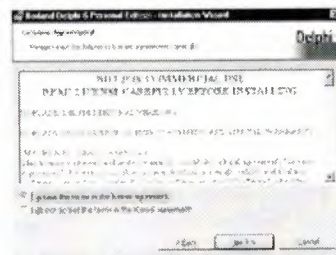
**Figure 1.1** The Select Page For Start Installation

While the setup runs, you'll need to enter your serial number and the authorization key (the two you got from inside a CdRom driver).



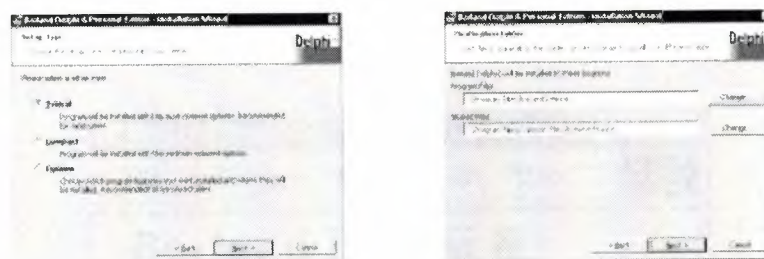
**Figure 1.2** Serial Number And Authorization Screen

Later, the License Agreement screen wills popup:



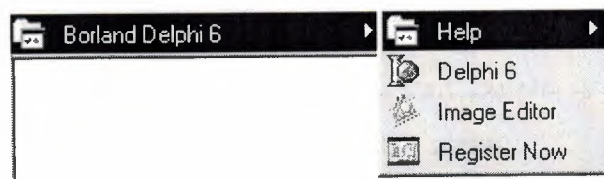
**Figure 1.3** License Agreement Screen

After that, you have to pick the Setup Type, choose Typical. This way Delphi 6 Enterprise will be installed with the most common options. The next screen prompts you to choose the Destination folder.



**Figure 1.4** SetUp Type and Destination Folder Screen

At the end of the installation process, the set-up program will create a sub menu in the Programs section of the Start menu, leading to the main Delphi 6 Enterprise program plus some additional tools.



**Figure 1.5** Start Menu

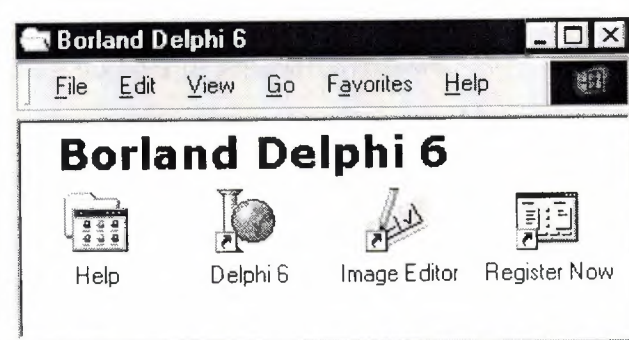
## 1.4 A Tour of the Environment

This chapter explains how to start Delphi and gives you a quick tour of the main parts and tools of the Integrated Development Environment (IDE)

### 1.4.1 Running Delphi for the First Time

You can start Delphi in a similar way to most other Windows applications:

- Choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu
- Choose Run from the Windows Start menu and type Delphi32
- Double-click Delphi32.exe in the \$(DELPHI)\Bin folder. Where \$(DELPHI) is a folder where Delphi was installed. The default is C:\Program Files\Borland\Delphi6.
- Double-click the Delphi icon on the Desktop (if you've created a shortcut)



**Figure 1.6** Borland Delphi 6 Folder



### 1.4.2 The Delphi IDE

As explained before, one of the ways to start Delphi is to choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu.

When Delphi starts (it could even take one full minute to start – depending on your hardware performance) you are presented with the IDE: the user interface where you can design, compile and debug your Delphi projects.

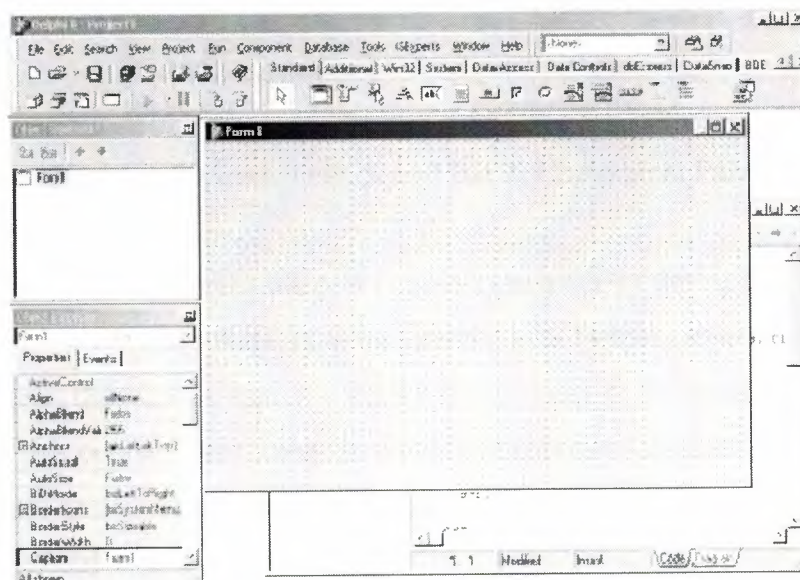


Figure 1.7 IDE

Like most other development tools (and unlike other Windows applications), Delphi IDE comprises a number of separate windows.

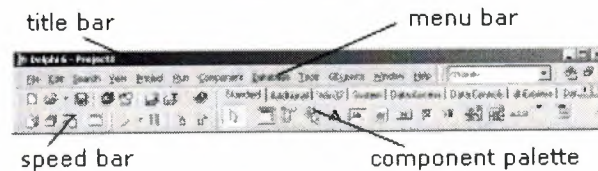
Some of the facilities that are included in the “Integrated Development Environment” (IDE) are listed below:

- A syntax sensitive program file editor
- A rapid optimizing compiler
- Built in debugging /tracing facilities
- A visual interface developer
- Syntax sensitive help files
- Database creation and editing tools

- Image/Icon/Cursor creation / editing tools
- Version Control CASE tools

### 1.4.3 The Menus & Toolbar

The main window, positioned on the top of the screen, contains the main menu, toolbar and Component palette.



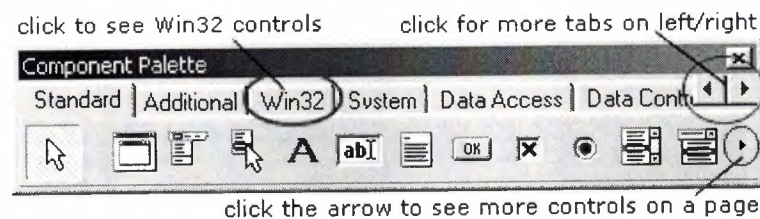
**Figure 1.8** Menu, Title, Speed Bar & Component Palette

The title bar of the main window contains the name of the current project (you'll see in some of the future chapters what exactly is a Delphi project). The menu bar includes a dozen drop-down menus – we'll explain many of the options in these menus later through this course. The toolbar provides a number of shortcuts to most frequently used operations and commands – such as running a project, or adding a new form to a project. To find out what particular button does, point your mouse “over” the button and wait for the tool tip. As you can see from the tool tip (for example, point to [Toggle Form/Unit]), many tool buttons have keyboard shortcuts ([F12]).

The menus and toolbars are freely customizable. I suggest you to leave the default arrangement while working through the chapters of this course.

### 1.4.4 The Component Palette

You are probably familiar with the fact that any window in a standard Windows application contains a number of different (visible or not to the end user) objects, like: buttons, text boxes, radio buttons, check boxes etc. In Delphi programming terminology such objects are called controls (or components). Components are the building blocks of every Delphi application. To place a component on a window you drag it from the component palette. Each component has specific attributes that enable you to control your application at design and run time.



**Figure 1.9** Component Palates

Depending on the version of Delphi (assumed Delphi 6 Personal through this course), you start with more than 85 components at your disposal – you can even add more components later (those that you create or from a third party component vendor).

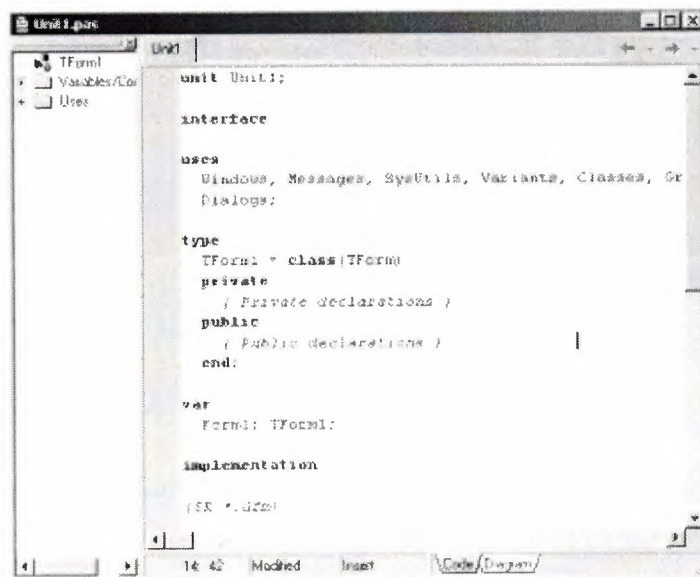
The components on the Component Palette are grouped according to the function they perform. Each page tab in the Component palette displays a group of icons representing the components you can use to design your application interface. For example, the Standard and Additional pages include controls such as an edit box, a button or a scroll box.

To see all components on a particular page (for example on the Win32 page) you simply click the tab name on the top of the palette. If a component palette lists more components that can be displayed on a page an arrow will appear on a far right side of the page allowing you to click it to scroll right. If a component palette has more tabs (pages) that can be displayed, more tabs can be displayed by clicking on the arrow buttons on the right-hand side.

#### **1.4.5 The Code Editor**

Each time you start Delphi, a new project is created that consists of one \*empty\* window. A typical Delphi application, in most cases, will contain more than one window – those windows are referred to as forms.

In our case this form has a name, it is called Form1. This form can be renamed, resized and moved, it has a caption and the three standard buttons which are minimize, maximize and close. As you can see a Delphi form is a regular Windows window



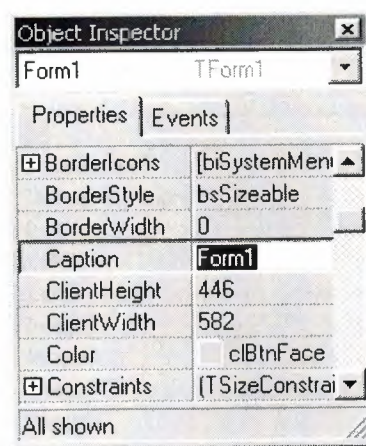
**Figure 1.10** Code Editor Window

If the Form1 is the active window and you press [F12], the Code Editor window will be placed on top. As you design user interface of your application, Delphi automatically generates the underlying Object Pascal code. More lines will be added to this window as you add your own code that drives your application. This window displays code for the current form (Form1); the text is stored in a (so-called) unit – Unit1. You can open multiple files in the Code Editor. Each file opens on a new page of the Code editor, and each page is represented by a tab at the top of the window.

#### **1.4.6 The Object Inspector**

Each component and each form has a set of properties – such as color, size, position, caption – that can be modified in the Delphi IDE or in your code, and a collection of events – such as a mouse click, keypress, or component activation – for which you can specify some additional behavior. The Object Inspector displays the properties and events (note the two tabs) for the selected component and allows you to change the property value or select the response to some event.





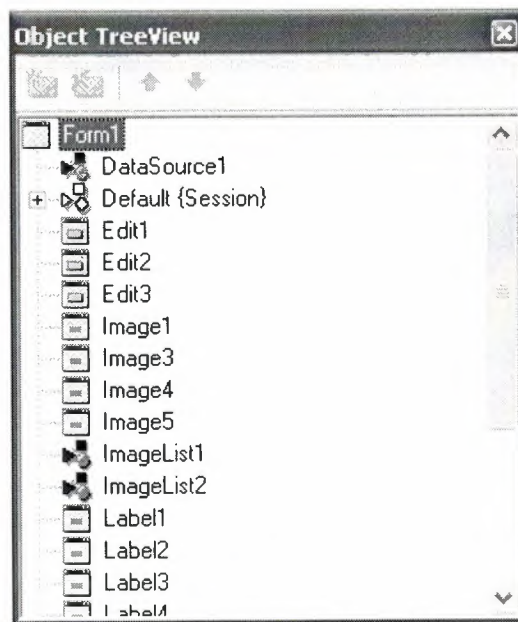
**Figure 1.11** Object Inspector

For example, each form has a Caption (the text that appears on it's title bar). To change the captions of Form1 first activate the form by clicking on it. In the Object Inspector find the property Caption (in the left column), note that it has the 'Form1' value (in the right column). To change the captions of the form simply type the new text value, like 'My Form' (without the single quotes). When you press [Enter] the caption of the form will change to My Form.

Note that some properties can be changed more simply, the position of the form on the screen can be set by entering the value for the Left and Top properties – or the form can be simply dragged to the desired location.

#### **1.4.7 The Object TreeView**

Above the Object Inspector you should see the Object TreeView window. For the moment its display is pretty simple. As you add components to the form, you'll see that it displays a component's parent-child relationships in a tree diagram. One of the great features of the Object TreeView is the ability to drag and drop components in order to change a component container without losing connections with other components.



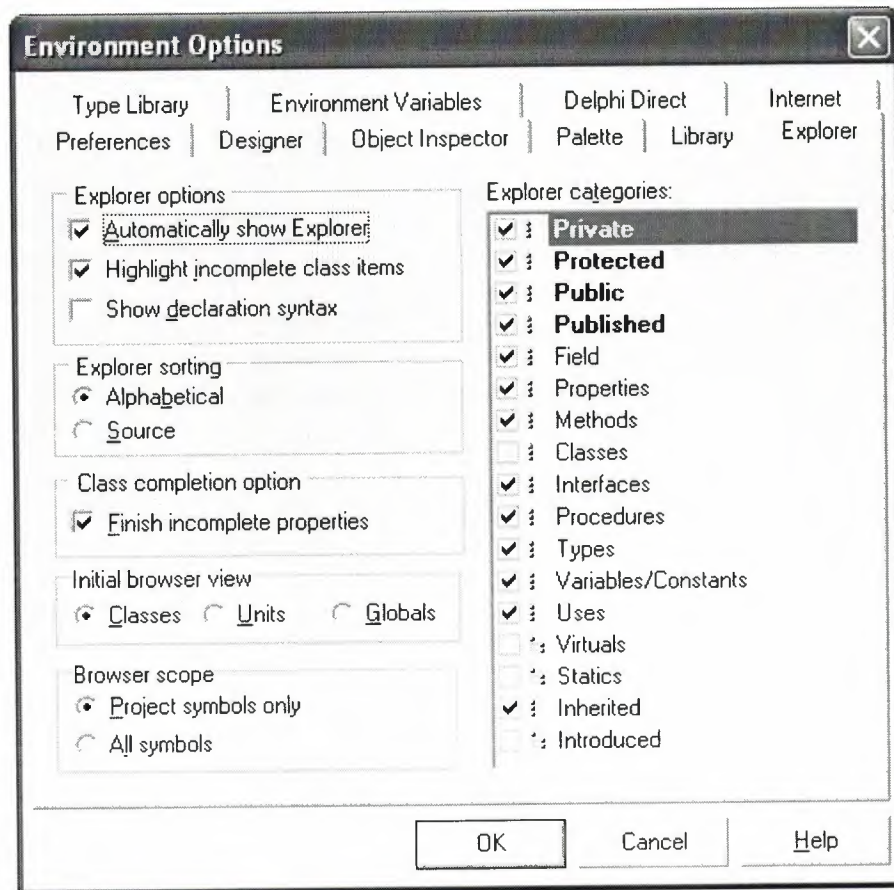
**Figure 1.12** Object Tree View

The Object TreeView, Object Inspector and the Form Designer (the Form1 window) work cooperatively. If you have an object on a form (we have not placed any yet) and click it, its properties and events are displayed in the Object Inspector and the component becomes focused in the Object TreeView.

#### **1.4.8 Class Completion**

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press Ctrl+Shift+C, or right-click and select Complete Class at Cursor. Delphi automatically adds private read and write specifies to the declarations for any properties that require them, and then creates skeleton code for all the class's methods. You can also use Class Completion to fill in class declarations for methods you've already implemented.

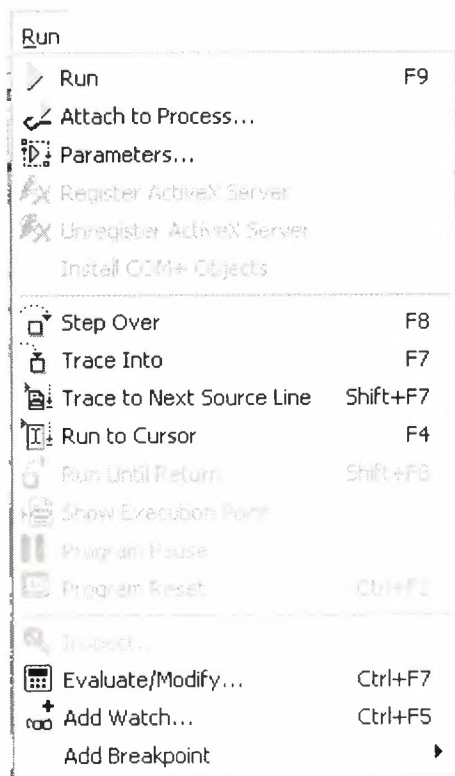
To configure Class Completion, choose Tools | Environment Options and click the Explorer tab.



**Fig.1.13** Class Completion

#### 1.4.9 Debugging applications

The IDE includes an integrated debugger that helps you locate and fix errors in your code. The debugger lets you control program execution, watch variables, and modify data values while your application is running. You can step through your code line by line, examining the state of the program at each breakpoint.



**Figure1.14 Run**

To use the debugger, you must compile your program with debug information. Choose Project | Options, select the Compiler page, and check Debug Information. Then you can begin a debugging session by running the program from the IDE. To set debugger options, choose Tools | Debugger Options.

Many debugging windows are available, including Breakpoints, Call Stack, Watches, Local Variables, Threads, Modules, CPU, and Event Log. Display them by choosing View | Debug Windows. To learn how to combine debugging windows for more convenient use, see “Docking tool windows”.

#### **1.4.10 Exploring Databases**

The SQL Explorer (or Database Explorer in some editions of Delphi) lets you work directly with a remote database server during application development. For example, you can create, delete, or restructure tables, and you can import constraints while you are developing a database application.



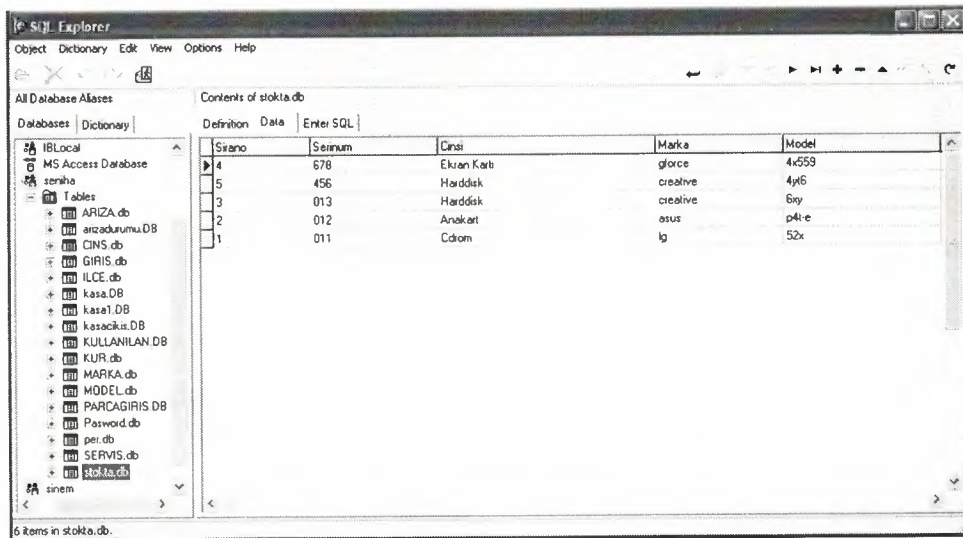


Figure 1.15 SQL Explorer

#### 1.4.11 Templates and the Object Repository

The Object Repository contains forms, dialog boxes, data modules, wizards, DLLs, sample applications, and other items that can simplify development. Choose File | New to display the New Items dialog when you begin a project. Check the Repository to see if it contains an object that resembles one you want to create.

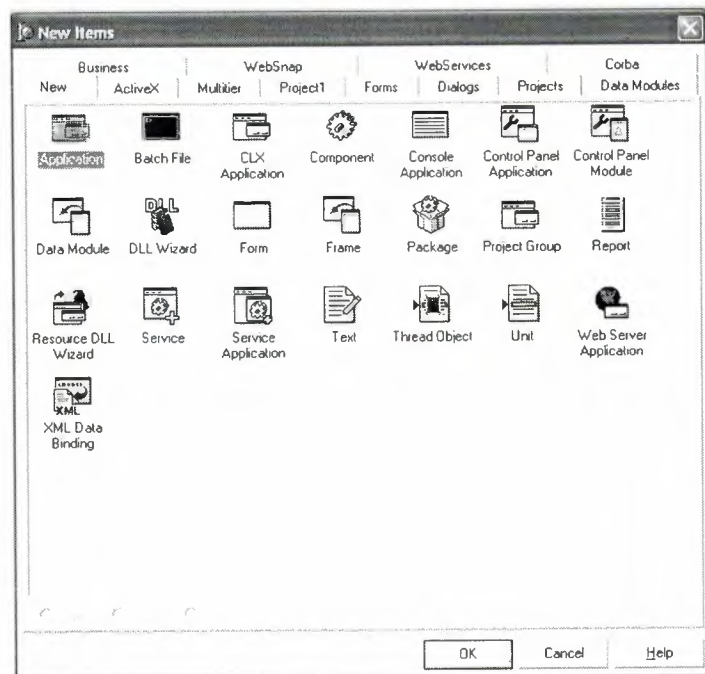


Figure 1.16 New Item

You can add your own objects to the Repository to facilitate reusing them and sharing them with other developers. Reusing objects lets you build families of applications with common user interfaces and functionality; building on an existing foundation also reduces development time and improves quality. The Object Repository provides a central location for tools that members of a development team can access over a network.

## **1.5 Programming with Delphi**

The following section provides an overview of software development with Delphi.

### **1.5.1 Starting a New Application**

Before beginning a new application, create a folder to hold the source files.

1. Create a folder in the Projects directory off the main Delphi directory.
2. Open a new project.

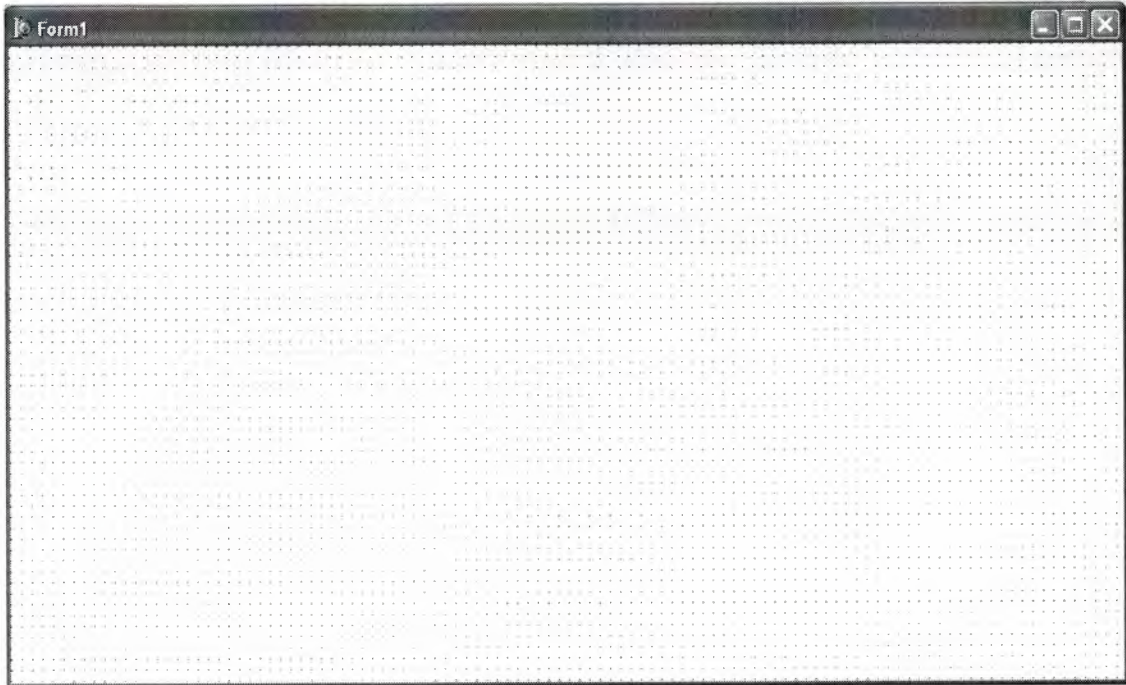
Each application is represented by a project. When you start Delphi, it opens a blank project by default. If another project is already open, choose File | New Application to create a new project. When you open a new project, Delphi automatically creates the following files.

- Project1.DPR : a source-code file associated with the project. This is called a project file.
  - Unit1.PAS : a source-code file associated with the main project form. This is called a unit file.
  - Unit1.DFM : a resource file that stores information about the main project form. This is called a form file.
3. Choose File | Save All to save your files to disk. When the Save dialog appears, navigate to your folder and save each file using its default name.

Later on, you can save your work at any time by choosing File | Save All.

When you save your project, Delphi creates additional files in your project directory. You don't need to worry about them but don't delete them.

When you open a new project, Delphi displays the project's main form, named Form1 by default. You'll create the user interface and other parts of your application by placing components on this form.



**Figure 1.17** Form Screen

The default form has maximize, minimize buttons and a close button, and a control menu

Next to the form, you'll see the Object Inspector, which you can use to set property values for the form and components you place on it.

The drop-down list at the top of the Object Inspector shows the current selected object. When an object is selected the Object Inspector shows its properties.

### **1.5.2 Setting Property Values**

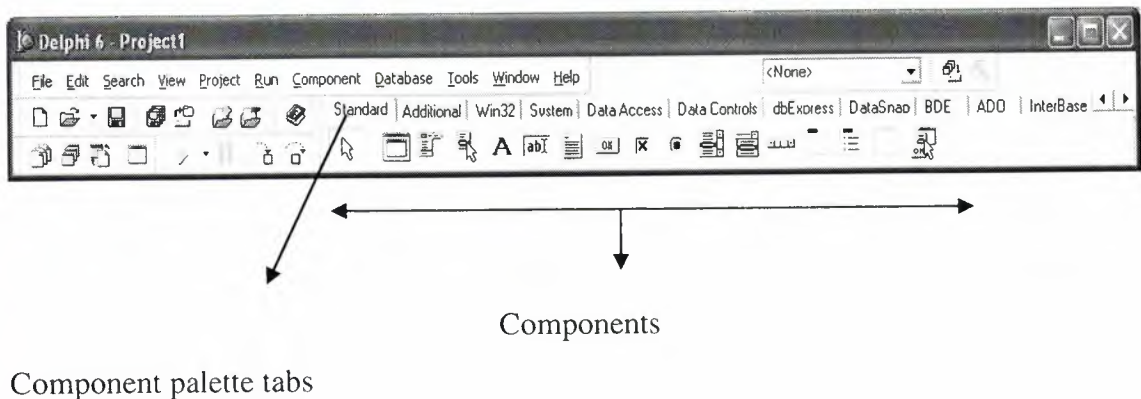
When you use the Object Inspector to set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called design-time settings.

For Example; set the background color of Form1 to Aqua.

Find the form's Color property in the Object Inspector and click the drop-down list displayed to the right of the property. Choose clAqua from the list.

### 1.5.3 Adding objects to the form

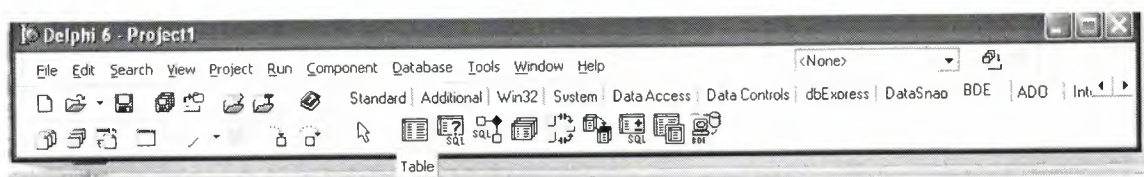
The Component palette represents components by icons grouped onto tabbed pages. Add a component to a form by selecting the component on the palette, then clicking on the form where you want to place it. You can also double-click a component to place it in the middle of the form.



**Figure 1.18** Standard Bar

### 1.5.4 Add a Table and a StatusBar to the Form

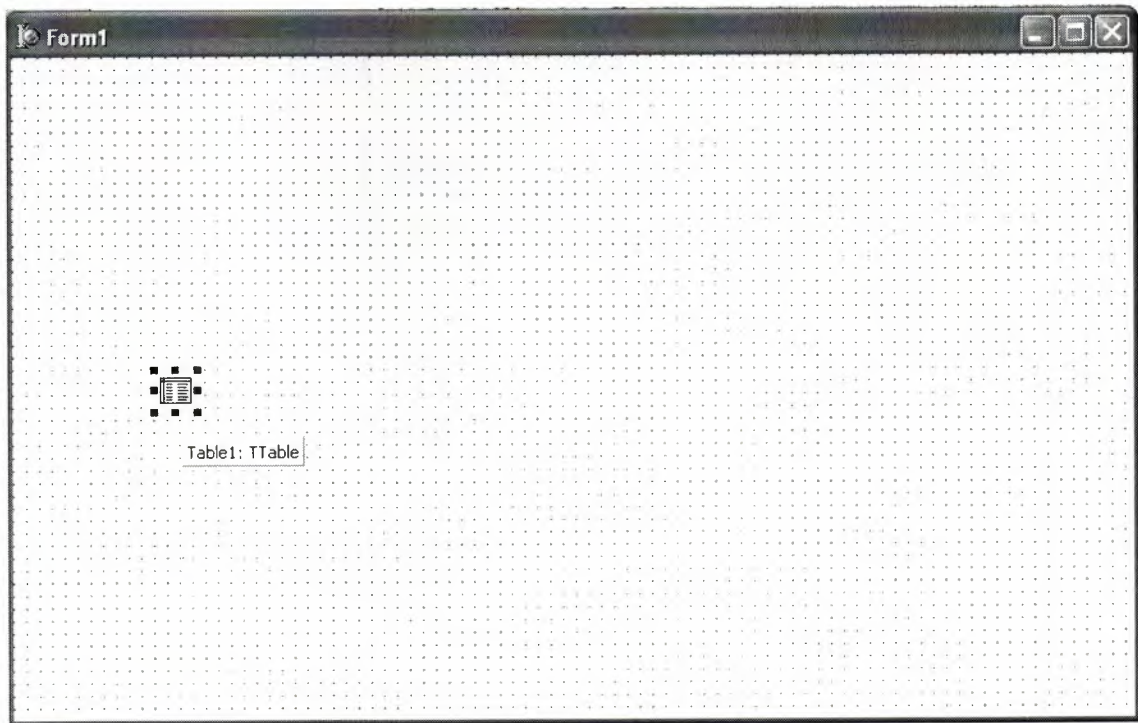
Drop a Table component onto the form. Click the BDE tab on the Component palette. To find the Table component, point at an icon on the palette for a moment; Delphi displays a Help hint showing the name of the component.



**Figure 1.19** BDE Component palette



When you find the Table component, click it once to select it, and then click on the form to place the component. The Table component is non visual, so it doesn't matter where you put it. Delphi names the object Table1 by default. (When you point to the component on the form, Delphi displays its name—Table1—and the type of object it is—Table.)

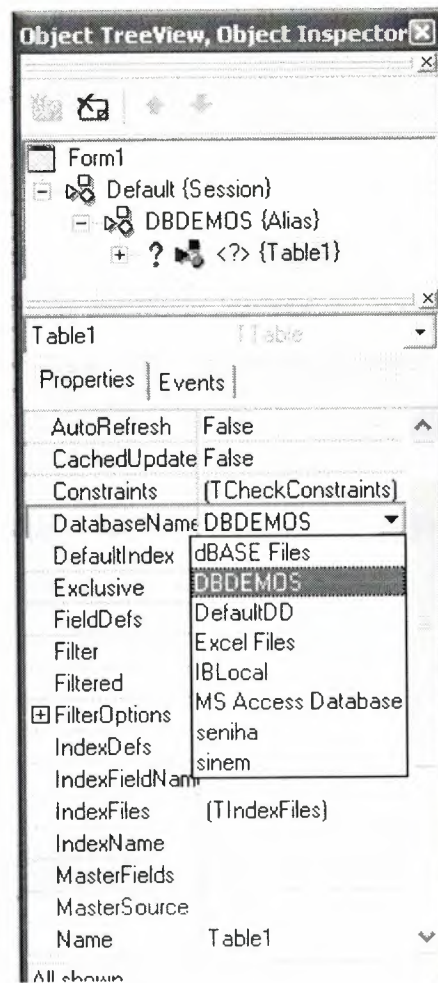


**Figure 1.20** Table in the Form

Each Delphi component is a class; placing a component on a form creates an instance of that class. Once the component is on the form, Delphi generates the code necessary to construct an instance object when your application is running.

Set the DatabaseName property of Table1 to DBDEMOS. (DBDEMOS is an alias to the sample database that you're going to use.)

Select Table1 on the form, and then choose the DatabaseName property in the Object Inspector. Select DBDEMOS from the drop-down list.



**Figure 1.21** Select DatabaseName

Double-click the StatusBar component on the Win32 page of the Component palette. This adds a status bar to the bottom of the application.

Set the AutoHint property of the status bar to True. The easiest way to do this is to double-click on False next to AutoHint in the Object Inspector. (Setting AutoHint to True allows Help hints to appear in the status bar at runtime.)

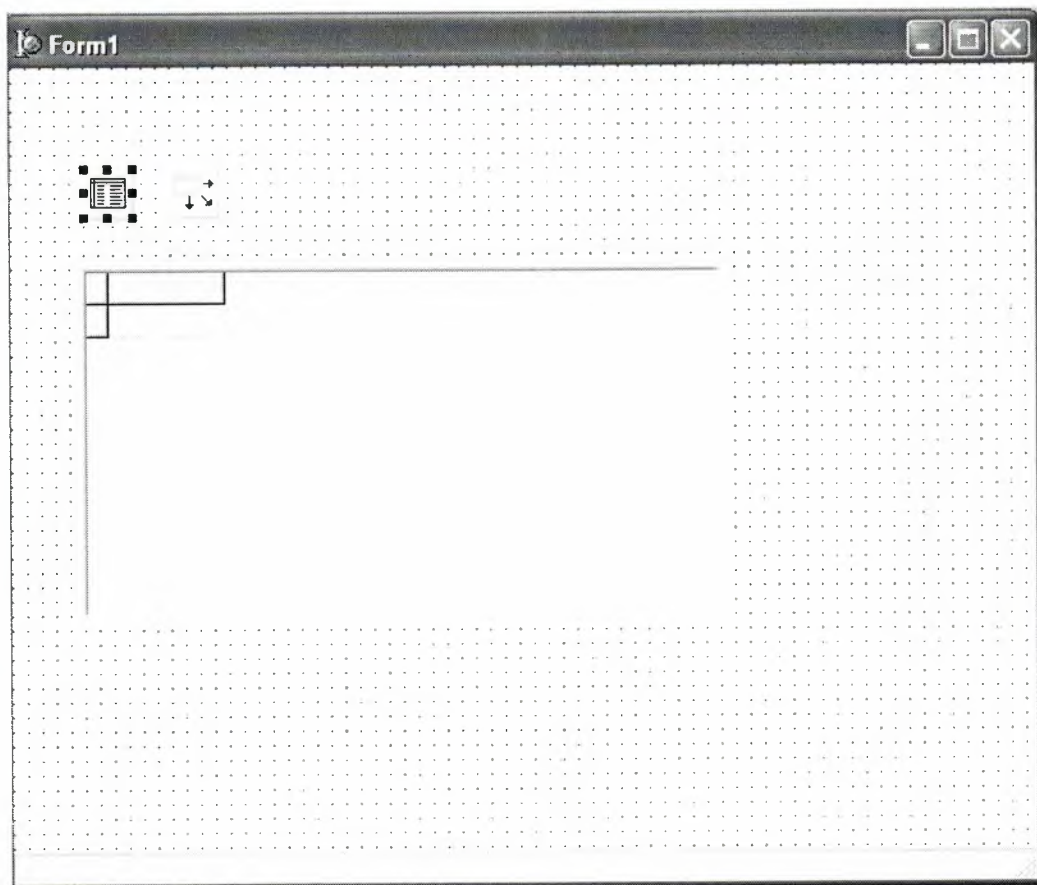
### 1.5.5 Connecting to a Database

The next step is to add database controls and a DataSource to your form.

1. From the Data Access page of the Component palette, drop a DataSource component onto the form. The DataSource component is non visual, so it doesn't matter where you put it on the form. Set its DataSet property to Table1.
2. From the Data Controls page, choose the DBGrid component and drop it onto your form. Position it in the lower left corner of the form above the status bar, and then expand it by dragging its upper right corner.

If necessary, you can enlarge the form by dragging its lower right corner. Your form should now resemble the following figure:

The Data Control page on Component palette holds components that let you view database tables.



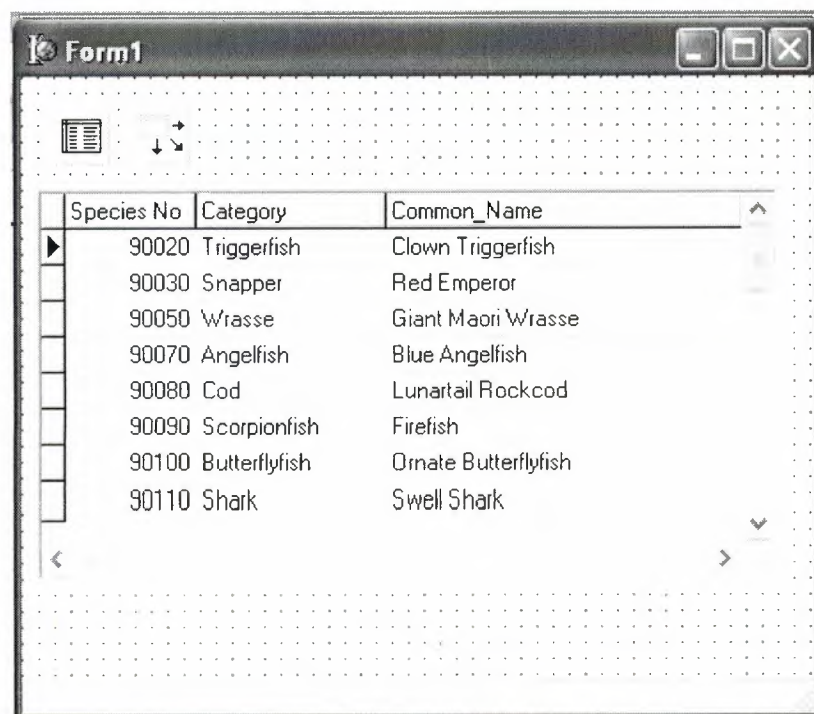
**Figure 1.22** DBGrid in the Form

3. Set DBGrid properties to align the grid with the form. Double-click Anchors in the Object Inspector to display `akLeft`, `akTop`, `akRight`, and `akBottom`; set them all to true.
4. Set the `DataSource` property of DBGrid to `DataSource1` (the default name of the `DataSource` component you just added to the form).

Now you can finish setting up the `Table1` object you placed on the form earlier.

5. Select the `Table1` object on the form, and then set its `TableName` property to `BIOLIFE.DB`. (Name is still `Table1`.) Next, set the `Active` property to `True`.

When you set `Active` to `True`, the grid fills with data from the `BIOLIFE.DB` database table. If the grid doesn't display data, make sure you've correctly set the properties of all the objects on the form, as explained in the instructions above. (Also verify that you copied the sample database files into your `...\Borland Shared\Data` directory when you installed Delphi.)



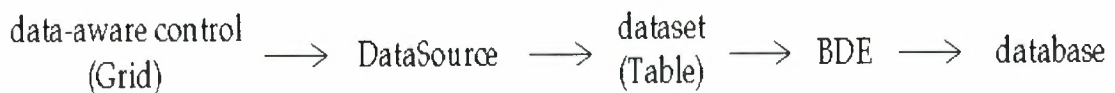
Species No	Category	Common_Name
90020	Triggerfish	Clown Triggerfish
90030	Snapper	Red Emperor
90050	Wrasse	Giant Maori Wrasse
90070	Angelfish	Blue Angelfish
90080	Cod	Lunartail Rockcod
90090	Scorpionfish	Firefish
90100	Butterflyfish	Ornate Butterflyfish
90110	Shark	Swell Shark

**Figure 1.23** Show Table



The DBGrid control displays data at design time, while you are working in the IDE. This allows you to verify that you've connected to the database correctly. You cannot, however, edit the data at design time; to edit the data in the table, you'll have to run the application.

6. Press F9 to compile and run the project. (You can also run the project by clicking the Run button on the Debug toolbar, or by choosing Run from the Run menu.)
7. In connecting our application to a database, we've used three components and several levels of indirection. A data-aware control (in this case, a DBGrid) points to a DataSource object, which in turn points to a dataset object (in this case, a Table). Finally, the dataset (Table1) points to an actual database table (BIOLIFE), which is accessed through the BDE alias DBDEMOS. (BDE aliases are configured through the BDE Administrator.)



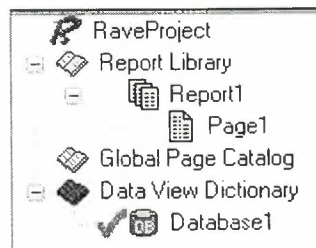
This architecture may seem complicated at first, but in the long run it simplifies development and maintenance. For more information, see “Developing database applications” in the Developer’s Guide or online Help.

## CHAPTER TWO

### 2 THE RAVE REPORTING

#### 2.1 Project Tree

The Project Tree provides an efficient way to visually manage all of the reports in your project. It quickly tells you the structure of your reporting project and the types of components contained on each page with icons that are the same as the component buttons. The Project Tree also visually shows parent-child relationships, the print order of component as well as the current selection (green check marks). You can select components by clicking on the component on the Page in the Visual Designer or on the Project Tree. Non-visual components appear only in the Project Tree in order not to clutter up your report design.



**Figure 2.1** Project Tree

There are three main sections in the Project Tree:

- The Report Library
- The Global Page Catalog
- The Data View Dictionary

Reports themselves can contain any number of page definitions. Global Pages are used to hold items that you want accessible to multiple reports. Data Views contain your field definitions and provide a link to the data in your application.

## 2.2 Design Tools

Rave is all about easy management. Besides making reporting easy and organized, Rave likes to keep itself organized and all according to what you want.

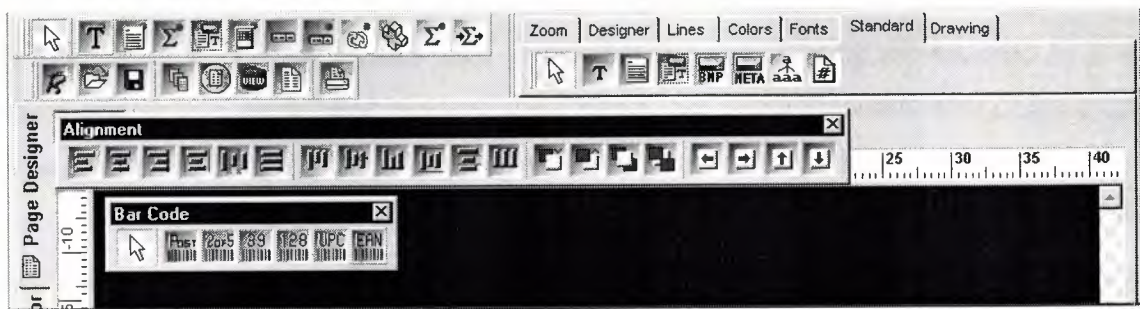


Figure 2.2 Toolbars

Since Rave is designed to be of ease to you there are three easy three ways for you to manage the many toolbars within Rave, which are:

- Tab-docking
- Normal docking
- Free-floating

Rave's many toolbars make it easy to design even the most complicated report. The toolbars include: Project, Designer, Zoom, Alignment, Color, Line, Font, Standard, Drawing, Report and Barcode component toolbars. Since it is possible to create and install new components, you may have other component toolbar buttons in your designer.



Figure 2.3 Project Toolbar

The Project toolbar provides quick access to project level functions such as New Project, Project Open, Project Save, New Report, New Global Page, New Data View, New Report Page or Execute Report.



**Figure 2.4** Designer Toolbar

The Designer toolbar allows you to change the characteristics of the Page in the Visual Designer. Characteristics such as whether the grid is being shown, snap to grid, draw grid on top, show band headers, show rulers, and show the waste area of the page. The last button brings up Rave's extensive Preferences dialog, which is described later.



**Figure 2.5** Zoom Toolbar

When you are working on a report with a complex design, you will find it much easier if you become familiar with the Zoom toolbar, which gives you quick access to Rave's extensive zooming capabilities. Select the zoom percent from a drop down list, type it in or use the Zoom Tool, Zoom In, Zoom Out, Zoom Selected, Zoom Page Width or Zoom Whole Page buttons.



**Figure 2.6** Alignment Toolbar

To help keep your report looking professional, Rave's Alignment toolbar provides access to a whole host of options to micro-manage the components on your page. The Left/Top, Center, Right/Bottom, Center In Parent, Space Equally, Equate Widths/Heights options offer the traditional alignment options. The Move Forward, Move Behind, Bring to Front and Send to Back order movement buttons allow you to change the print order of components and are visually backed up by the listing of the components in the Project Tree. Lastly, the buttons Tap Left, Tap Right, Tap Up and

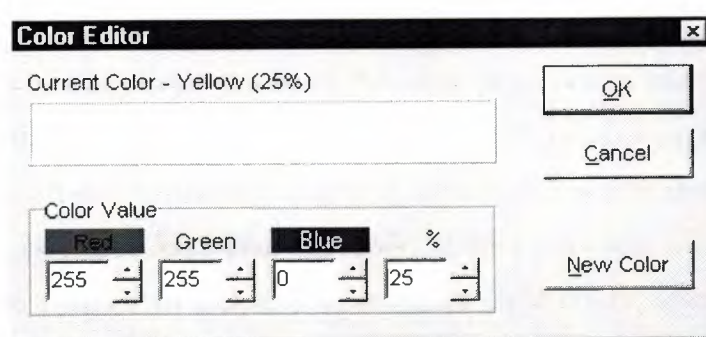


Tap Down allow you to micro-adjust the position of components to the exact position you need.



**Figure 2.7** Colors Toolbar

The Color toolbar allows you to quickly select the primary and secondary colors of your components. There are 8 color spots that you can use to store any custom colors that you will be reusing throughout the project. If the colors available aren't enough, you can double click on the custom color palettes and create a different color using Rave's Color Editor (shown at right). With the Color Editor, you can select from a wider variety of colors or create your own combination of Red, Green and Blue and even select a percent saturation for the current color.



**Figure 2.8** Colors Editor

The Line toolbar is a useful tool for changing the line/border thickness and style for components such as Line and Circle. Sizes are listed in points instead of pixels so that your lines will always be the same thickness on your reports no matter the resolution of the printer that you are using.



**Figure 2.9** Line Toolbar

The Font toolbar provides quick access to a text component's font and alignment properties. It can also be useful for quickly viewing the font options for the currently selected text component(s).



Figure 2.10 Fonts Toolbar

## 2.3 Reuse and Maintenance Tools

Reports often take a large part of the development time for an application. Many times, there are many similarities between the design of separate reports.

This is where Rave's Mirroring technology comes in. When a component is set to mirror another, it assumes the appearance and properties of the component it is mirroring. The two components can be on the same page, across pages within the same report or on a global page. This is the primary purpose of a global page. You can almost think of it like an Object Repository, a central location for you to store reporting items that you want accessible to more than one report. If the component is a container control like TraveSection (similar to Delphi's Tpanel), all child components are mirrored as well. When the original component changes, all mirroring components will also change. While the mirrored component cannot change its properties, you can add additional components if it is a container control.

Here are just a few examples of where Mirroring would be useful:

Your customer wants a standard page header and footer on every page of their 50 reports. Now imagine you have all the reports done and your customer wants to change the layout of the headers and footers.

The Old Way – You would need to open up all 50 report definitions and change them one at a time.

The Rave Way – You would mirror the standard header and footer on each report you create and then any changes would only have to be done in one location. Also, if the standard header included a large bitmap, your reporting project would only contain a single copy rather than the many copies that a traditional report designer would require. You have to replicate a pre-printed form. The problem is there are 6 different variations of this form with only minor differences between each.

The Old Way – Assuming a traditional report designer could even handle this type of report, you would create the first form, cut and paste it into the second, make the minor modifications, then repeat for the other 4 forms, ending up with 6 reports that would be hard to maintain and take up a lot more memory.

The Rave Way – You would first create the common items of the form on a separate page, then mirror those on each form and add the unique parts for each as needed. If anything ever needed to be changed in the common section of the form, you would only need to change it in one place and since you're sharing most of the form's content, the report definitions take up much less room.

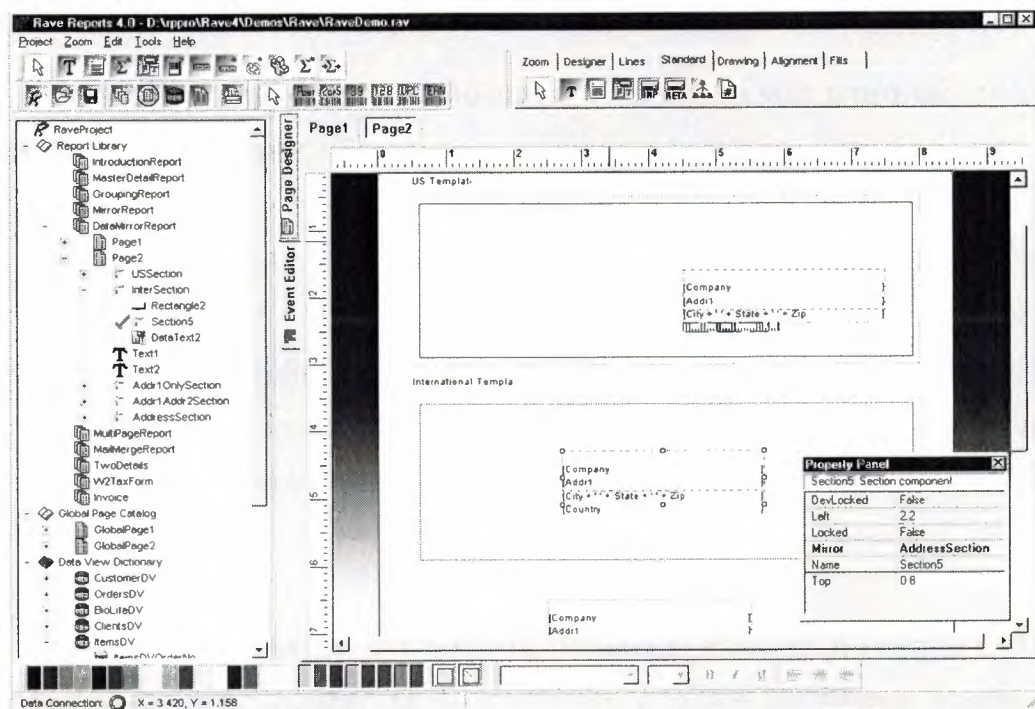


Figure 2.11 Mirror Report Example



Every text component has a FontMirror property which you can assign to a FontMaster component. This will allow you to change the fonts of many text controls from a single location. Imagine having Header, Body and Footer FontMaster components on a global page and changing the appearance of all of your reports with just a few mouse clicks.

Another important aspect of maintaining any large project is documentation. The Project and every Report, Page, Data View and Data Field component has a multi-line Description Property that can be used to comment the intended usage or other information. This can be useful if you are coming back to a project that you last worked on 6 months ago or especially if another programmer or your end user will be modifying reports that you created.

## 2.4 Standard Components



**Figure 2.12** Standard Tool Bar

**Text** – This component is used to display fixed text on your report for items such as column headers or report titles.

**Memo** – This component is used to display fixed text in a word wrapped fashion on your report. Using the MailMergeItems property and the Mail Merge Editor shown below, you can create a mail merge type of report where Rave will replace tokens in the memo text with a replacement string. Note that this replacement string can be edited with the Edit button, which will display the Data Text Editor for quite a bit of extra functionality.

**Section** – This component is a terrific component manager. It acts as a container for other components, in other words it help you to group components together. By



properly using section components and mirroring, you can create reusable and maintainable reports in no time flat.

**Bitmap** – This component is used to display a bitmap (\*.bmp). Through the FileLink property you can reference a file on the hard disk.

**MetaFile** – This component is used to display a metafile (\*.wmf). Through the FileLink property you can reference a file on the hard disk.

**FontMaster** – This component is used to control the font characteristics of any text control through their FontMirror properties. See Reuse and Maintenance for more information.

## 2.5 Drawing Components

**Line** – Draws a diagonal line. (This may not seem like a unique feature but did you know that most Delphi reporting tools cannot create a diagonal line visually.)



**Figure 2.13** Drawing Tool Bar

**Hline** – Draws a horizontal line.

**Vline** – Draws a vertical line.

**Rectangle** – Draws a rectangle.

**Square** – Draws a square.

**Ellipse** – Draws an ellipse.

**Circle** – Draws a circle.

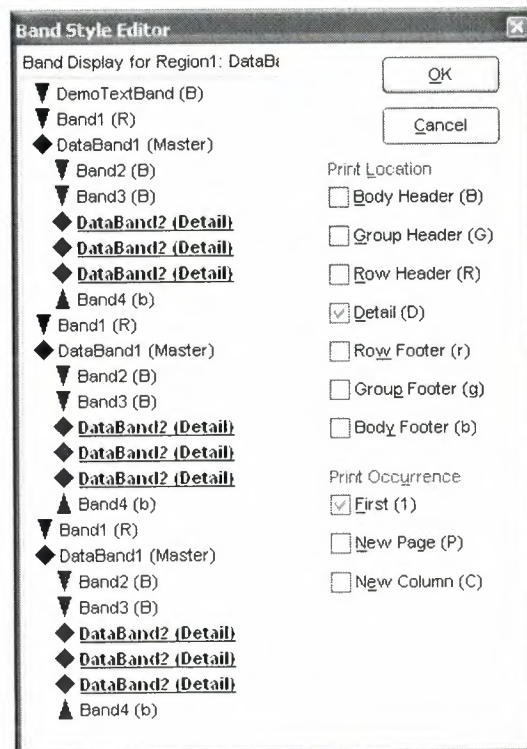
## 2.6 Reporting Components

**Region** – This component acts as a container for Band and DataBand components. To create a composite or sub-report, simply drop more than one region on a page and add the appropriate bands to each.



**Figure 2.14** Report Tool Bar

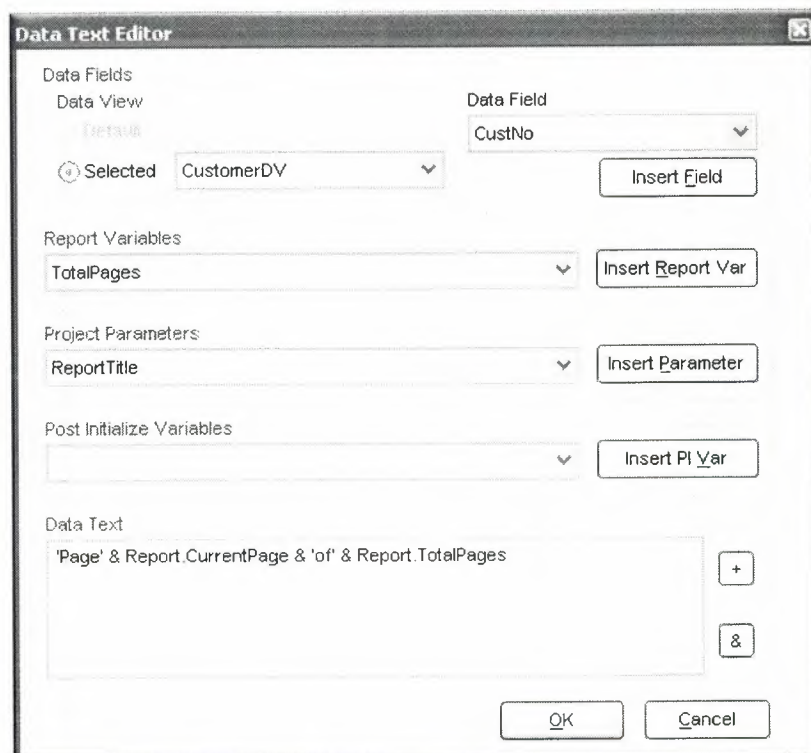
**Band** – This component is primarily used to create header and footer bands in a banded style report. A Band component can only be created within a region and its purpose is controlled through the Band Style Editor shown below. The Band Style Editor displays a virtual layout of all of your bands for the given print locations of each band or data band. Note that you can create as many Bands as you like and a Band may print in multiple locations if the report design requires it. So for example, if you want a solid horizontal line to appear above and below a detail body, you could create a single band and set it to print on both the Body Header and Body Footer. You can also control the Print Occurrence for a Band, having it continue on a new page or column or any combination of occurrence settings. You can set a Band to group on specific fields and can create as many different types of group headers or footers as your report requires. Basically, with Rave's Band and DataBand components, you'll be able to create just about any banded style layout that you can imagine.



**Figure 2.15** Band Style Editor

**DataBand** – The DataBand component is fairly similar to a band component except that it is tied to a particular DataView and iterates across the rows in the DataView. You can link DataBands together for Master-Detail to unlimited levels or multiple details on the same level. Some advanced features that are supported by a DataBand include KeepBodyTogether, KeepRowTogether, StartNewPage, MaxRows and Orphan/Widow control.

**DataText** – The DataText component is the primary means to output fields from your database. You can quickly select a specific DataView and DataField with Property Panel or use the Data Text Editor shown below to create any combination of string constants, data fields, report variables or project parameters. The & concatenation operator is the same as the + operator, except that it also inserts a space. Report Variables are items such as total pages or current date and time in a variety of formats. Project Parameters are custom variables that you create and initialize from your Delphi application. Project Parameters can be used for items such as user defined report titles, printing the current user name or other custom information.

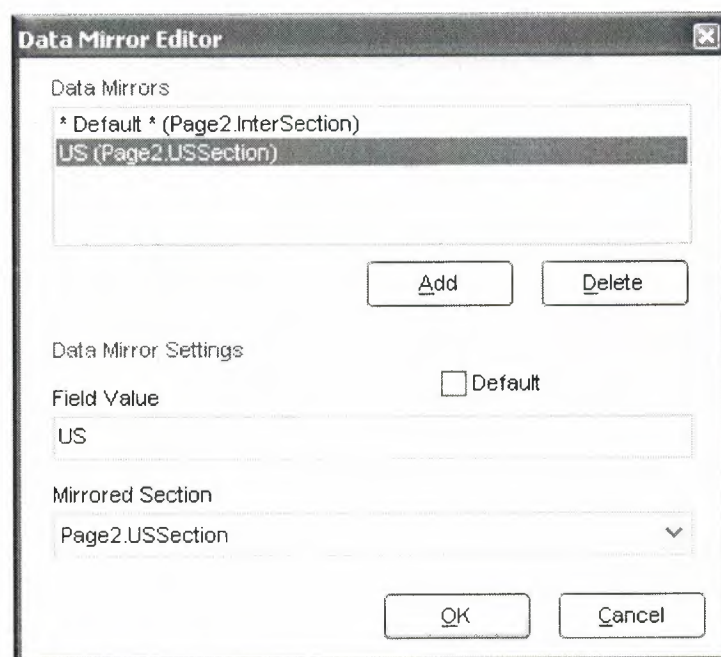


**Figure 2.16** Data Text Editor

**DataMemo** – This component is very similar to the Memo component except that it retrieves data from a DataField. DataMemo component's print text data out in a word wrapped fashion and the DataField can be any text type, not just memo fields. It also has RTF and mail merge support.

**CalcText** – This component is used to perform simple operations such as Sum, Average, Count, Min and Max on a data field. You can set the value as a running total and place it in any type of band or anywhere on the page) you need it.

**DataMirrorSection** – The data mirror section component is similar to Rave's section component (found in the Standard Toolbar) with one major difference, it will dynamically mirror another section depending upon the value of a DataField. You configure the data mirror section using the Data Mirror Editor (shown below). This component is very useful for printing out data that has different formats depending upon the type of data. One example is an address field that could print a US format if the country field is "US" and an international format otherwise (using the Default option in the Data Mirror Editor). You could also print Boolean field values with your own custom bitmaps.



**Figure 2.17** Data Mirror Editor



## 2.7 Barcode Components



Figure 2.18 Barcode Toolbar

PostNetBarCode – Prints a US PostNet bar code.

I2of5BarCode – Prints Interleaved 2 of 5 barcodes.

Code39BarCode – Prints standard and extended Code 39 barcodes.

Code128BarCode – Prints A, B and C Code 128 barcodes.

UPCBarCode – Prints UPC-12 barcodes.

EANBarCode – Prints EAN-13 barcodes.

## 2.8 Anchors

Anchors are a powerful way to create a report that dynamically adjusts to changing sizes. This allows you to create reports that can print well whether the user selects landscape or portrait, 8.5" by 11" or A4. There are 6 different anchor values for both the horizontal and vertical dimensions to allow you to control each component in exactly the manner that it needs. The Anchor Editor (shown at right) even shows you a helpful bitmap of how each anchor setting works.

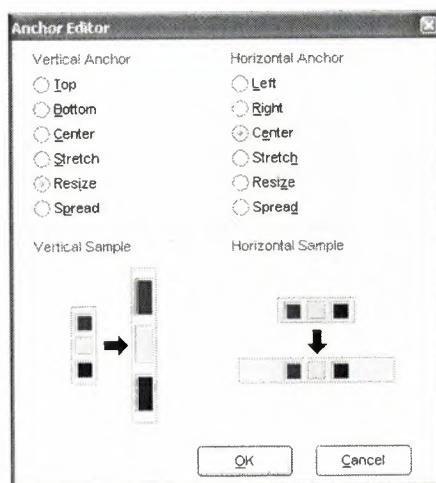


Figure 2.19 Anchor Editor

## 2.9 Code Based Reports

Lately Delphi has decided to include Rave Reports as the default reporting solution, replacing Quick Reports. Since they work in very different paradigms, many people were confused by the new environment. This is intended as an introduction for people who haven't worked with Rave yet, and would like to start.

Nowadays Delphi ships with Rave Reports 5.0.8. If you haven't already, download the update from the registered users page, since it fixes some important problems.

You can develop reports with Rave using two different ways: Code Based or with the Visual Designer.

With Code Based, you write reports using plain Delphi code. That provides a very flexible way displaying any kind of data, allowing any kind of complex layouts.

To write a code based report, just drop a TrvSystem component on the form and write the report on the OnPrint event handler. Sender is the report you are creating, and can be typecasted to TbaseReport. It contains all the methods you need to output information to that particular report.

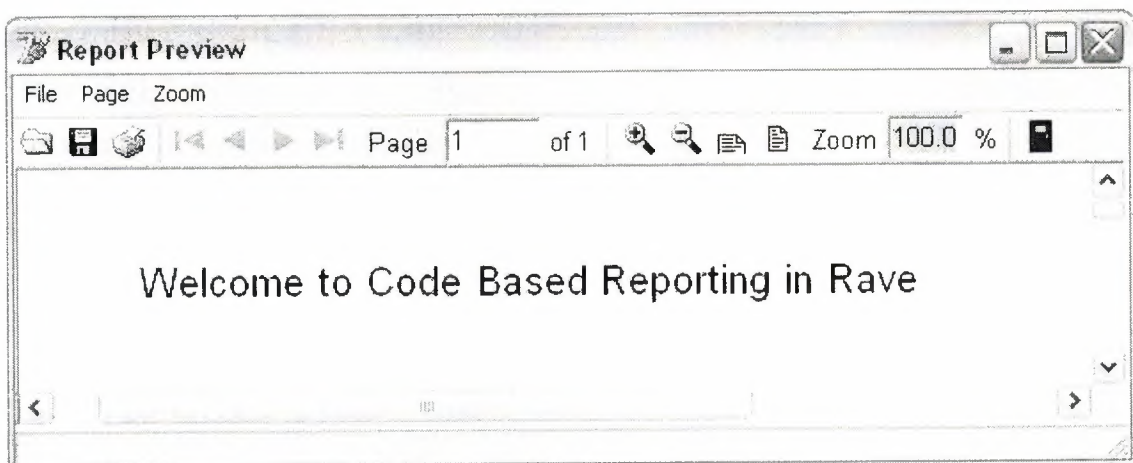
### 2.9.1 Simple Code Base Report

Here's a simple report using the code based mechanism:

```
procedure TFormMain.RvSystemPrint(Sender: Tobject);
begin
  with Sender as TbaseReport do
  begin
    SetFont('Arial', 15);
    GotoXY(1,1);
    Print('Welcome to Code Based Reporting in Rave');
  end;
end;
```

To execute this report, call `RvSystem.Execute` method.

So, what does that simple code do? First, it calls `SetFont` to select the font and size of the text that will be printed from that point on. Then it positions the cursor on the coordinates (1,1). These coordinates are expressed using the units set in the `SystemPrinter.Units` property of the `RvSystem` object, and it defaults to Inches. You can set it to `unUser` and set a number relative to Inches in the `SystemPrinter.UnitsFactor` property. For example, if `UnitsFactor` was set to 0.5 then 1 unit would correspond to half an inch. Finally, the code calls the `Print` method to output the text. Here's the output:



**Figure 2.20** Report Preview

### 2.9.2 Tabular Code Based Report

Here's another example. It displays a list of the folders in the root of the current drive, along with a recursive count of number of files and folder, and total size of the files included in each folder.

```
Procedure TFormMain.PrintTabularReport(Report: TbaseReport);  
var  
    FolderList : TstringList;  
    I          : Integer;  
    NumFiles   : Cardinal;
```

```

NumFolders : Cardinal;
SizeFiles : Cardinal;
Root      : string;
begin
  with Report do
  begin
    SetFont('Arial', 15);
    NewLine;
    PrintCenter('List of Folders in the Drive Root', 4);
    NewLine;
    NewLine;
    ClearTabs;
    SetTab(0.2, pjLeft, 1.7, 0, 0, 0);
    SetTab(1.7, pjRight, 3.1, 0, 0, 0);
    SetTab(3.1, pjRight, 3.5, 0, 0, 0);
    SetTab(3.5, pjRight, 4.5, 0, 0, 0);
    SetFont('Arial', 10);
    Bold := True;
    PrintTab('Folder Name');
    PrintTab('Number of Files');
    PrintTab('Number of Folders');
    PrintTab('Size of Files');
    Bold := False;
    NewLine;
    FolderList := TStringList.Create;
  try
    Root := IncludeTrailingPathDelimiter(ExtractFileDrive(ParamStr(0)));
    EnumFolders(FolderList, Root);
    for I := 0 to FolderList.Count - 1 do
    begin
      PrintTab(FolderList[I]);
      GetFolderInfo(IncludeTrailingPathDelimiter(Root+FolderList[I]),
        NumFiles, NumFolders, SizeFiles);
    end
  end
end

```

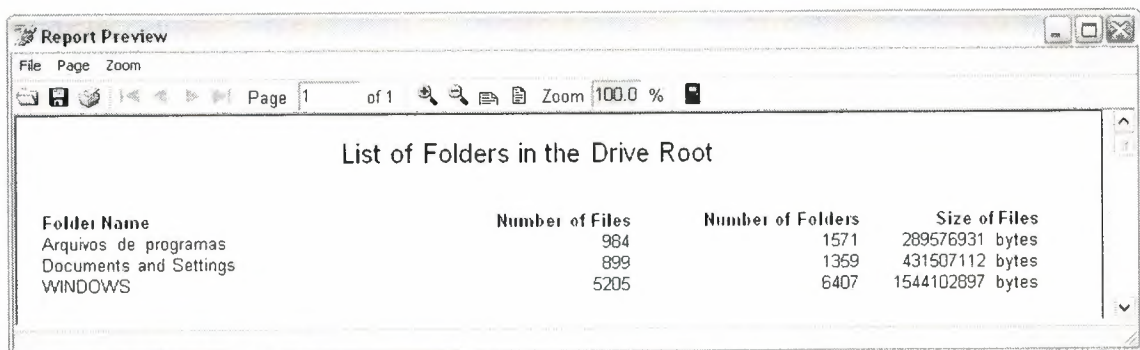


```

PrintTab(Format('%u',[NumFiles]));
PrintTab(Format('%u',[NumFolders]));
PrintTab(Format('%u bytes',[SizeFiles]));
NewLine;
end;
finally
  FolderList.Free;
end;
end;
end;
end;

```

Notice that a different approach has been taken: instead of specifying the coordinates of each text output, the printing was done using Lines and Columns as references. The line height depends on the size of the current font: each unit represents 1/72nds of an inch, so each line printed with a size 10 font will have, appropriately, a height of 0.138 inches. Lines are advanced after calls to PrintLn or NewLine. Columns are defined using calls to the SetTabs method, and the PrintTab method will print the text in the current column and advance to the next one. Here's the output:



The screenshot shows a window titled 'Report Preview' with a menu bar (File, Page, Zoom) and a toolbar. The main content area displays a table titled 'List of Folders in the Drive Root'. The table has four columns: 'Folder Name', 'Number of Files', 'Number of Folders', and 'Size of Files'. The data rows are: 'Arquivos de programas' (984 files, 1571 folders, 289576931 bytes), 'Documents and Settings' (899 files, 1359 folders, 431507112 bytes), and 'WINDOWS' (5205 files, 6407 folders, 1544102897 bytes).

Folder Name	Number of Files	Number of Folders	Size of Files
Arquivos de programas	984	1571	289576931 bytes
Documents and Settings	899	1359	431507112 bytes
WINDOWS	5205	6407	1544102897 bytes

**Figure 2.21** Report Preview

### 2.9.3 Graphical Code Based Report

You can include shapes and images in your code based report, along with the text. The following example demonstrates that:

```
procedure TFormMain.PrintGraphicsReport(Report: TbaseReport);
```

```
var
```

```

    Bitmap : Tbitmap;
begin
    with Report do
    begin
        Canvas.Brush.Color := clGray;
        Rectangle(0.3, 0.3, 4.7, 3.3);
        SetFont('Arial', 15);
        FontColor := clRed;
        PrintXY(0.5,0.5, 'Just look at all the graphics!');
        Bitmap := Tbitmap.Create;
    try
        Bitmap.LoadFromFile('delphi.bmp');
        PrintBitmap(3.5,0.3,1,1, Bitmap);
        PrintBitmap(1,2,3,3, Bitmap);
        Canvas.Pen.Color := clBlue;
        Canvas.Brush.Bitmap := Bitmap;
        Ellipse(5,0.3,6,3.3);
        Ellipse(2,1,4,1.9);
    finally
        Bitmap.Free;
    end;
    Canvas.Pen.Color := clBlack;
    Canvas.Brush.Style := bsSolid;
    Canvas.Brush.Color := clYellow;
    Pie(0.7,0.7,1.7,1.7,1,1,1,2);
    Canvas.Brush.Color := clGreen;
    Pie(0.7,0.7,1.7,1.7,1,2,1,1);
    end;
end;

```

In this example the methods Rectangle, Ellipse and Pie have been used draw shapes with different fills. Bitmaps were outputted using PrintBitmap and as the brush of the ellipses. Here's the output:

## Graphics Report Example

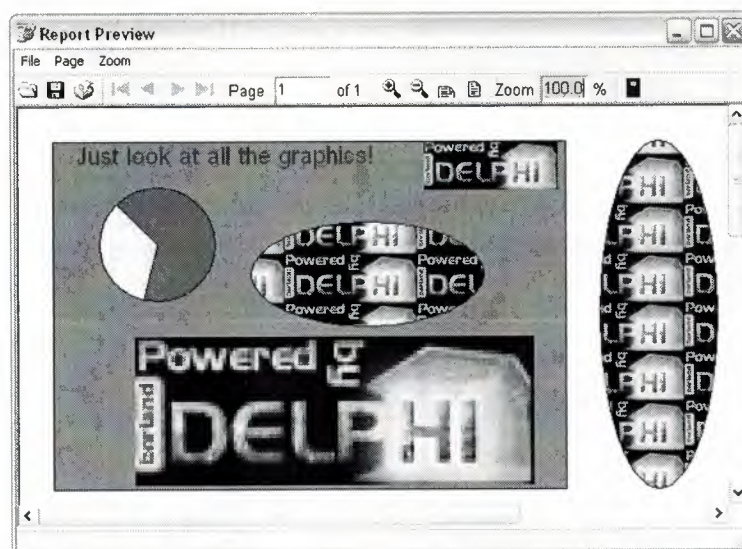


Figure 2.22 Report Preview

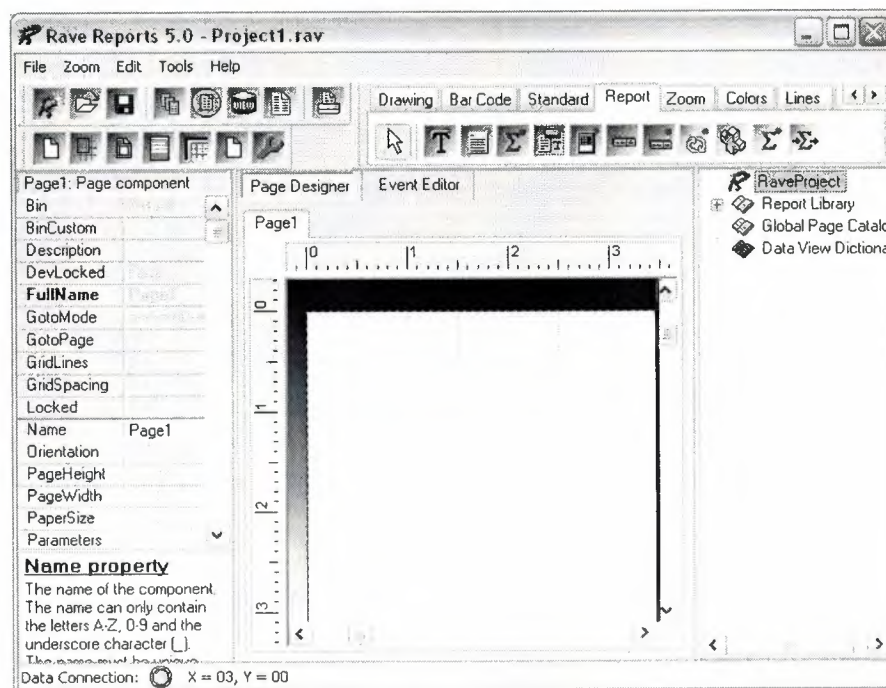
## 2.10 Visually Designed Reports

### 2.10.1 The Visual Designer

If you are used to work with Quick Reports, the default reporting engine included in the previous versions of Delphi, you created your reports using Delphi's own form designer, and they were save in the DFM, included as resources in your executable. Rave works a bit differently in this aspect: it has it's own report designer, and saves the report using it's own file format. This has some advantages, including the fact that your reports can be made "standalone", and be used or updated independently of your application, or even made available in a Intranet or in the Internet, using Nevrona's Rave Report Server. Of course, you can still have it saved in a form's DFM.

To get started with the Rave Visual Designer, drop a TrvProject in a form. This will be the link from your application to the reports you are developing. If you want, you can add a TrvSystem and link your RvProject to it, through it's Engine property. The RvSystem is the object responsible for the general configuration of the reports: the printer that is going to be used, the margins, the number of pages, and so on. To start a new project, double click the RvProject you added to the form, or select "Rave Visual Designer" from its context menu.

This is the interface that you will be working on:




**Figure 2.23** Rave Visual Designer

The interface is simple, and you might be familiar with some parts of it from Delphi's IDE. On the top there's the menu, the toolbar, and the component palette that contain the components that will be used in the reports. In the left there's the Object Inspector, which will be used to adjust the properties of the components of the report. In the middle there's the Page Designer or the Event Editor, and in the left there's the very usefull Project Treeview. For a quick overview of the components in the palette, you can go to Nevrona's Visual Designer page.

A Rave Project File can have one or more reports. That way you can keep common items between them in a single location, called Global Pages. If you expand the Report Library node of the Project Treeview, you can see that right now you are working on Report1. Clicking on it, its properties will show on the Inspector. Let's change it's name and call it SimpleReport. Next, go to the Standard tab on the Component Palette, and pick a Text component and add it to the page. Change its text property, and adjust its size and position. Here's how it looks like:



Anchor	(Top / Left)
Color	 Black
DevLocked	False
DisplayOn	doParent
<b>Font</b>	<b>Arial,15</b>
FontJustify	plLeft
FontMirror	
Left	1,000
Locked	False
Mirror	
Name	Text1
Rotation	0
Tag	0
<b>Text</b>	<b>Welcome to Rave Reports Visual I</b>
Top	1,000
<b>Truncate</b>	<b>True</b>
Width	4,000

**Figure 2.24** Component Palette: Standard Tab

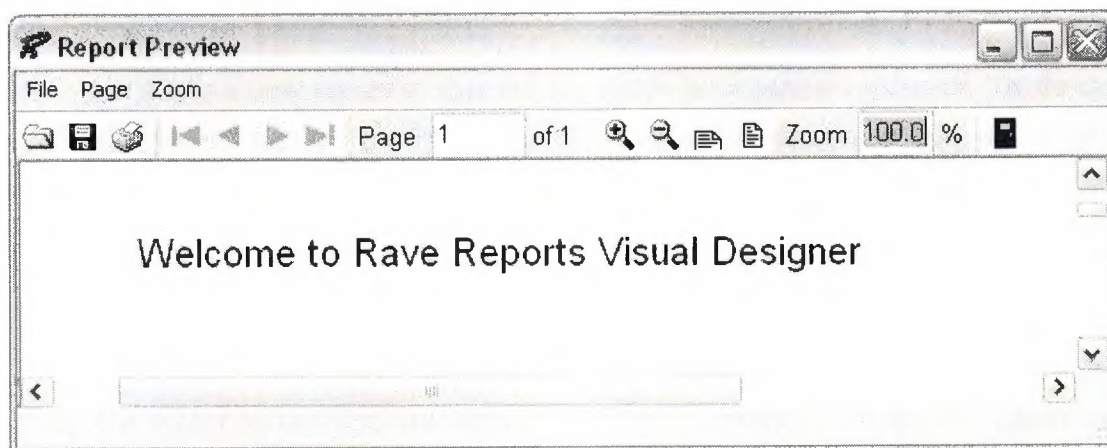
As you can see, the properties that were changed from the default values are shown in bold. In this case, I changed the Font, Text and Truncate properties. By default it does not highlight Name, Pos and Size changes. If you'd like to see them, right click the Inspector and uncheck "Exclude Name, Size and Pos changes" in the context menu.

You might have also noticed that Rave does not have an auto size property. You can use the Truncate property to have that effect: if truncate is false, the design time size will have no effect.

You can see the result of this simple report right on the designer: Press F9 or use File/Execute Report to run it. Now let's do it in our application. Save your project and return to Delphi. Change to ProjectFile property of RvProject to point to the file you just saved. To run the report, add a call to the Execute method of the RvProject object in a button click, for example.

RvProject.Execute will only work for now because we only have one report in this project. If we had multiple reports, we'd have to call SelectReport to choose one before calling Execute, or calling ExecuteReport directly.

Here's the output:



**Figure 2.25** Report Preview

Tip: If you Close and Open your project before executing, you won't need to recompile your application or restart it to see the changes you just made in the designer.

### **2.10.2 Interacting with the Project**

If you worked with Quick Reports, you might be used to manipulating the objects in runtime, changing their Position, Text and Visibility. After all, they were just Tobjects! While this is possible with Rave, and I'll cover it in a later article, it's a little harder than it was with QR. But don't worry, Rave provides a different answer to this kind of problems.

#### **Parameters**

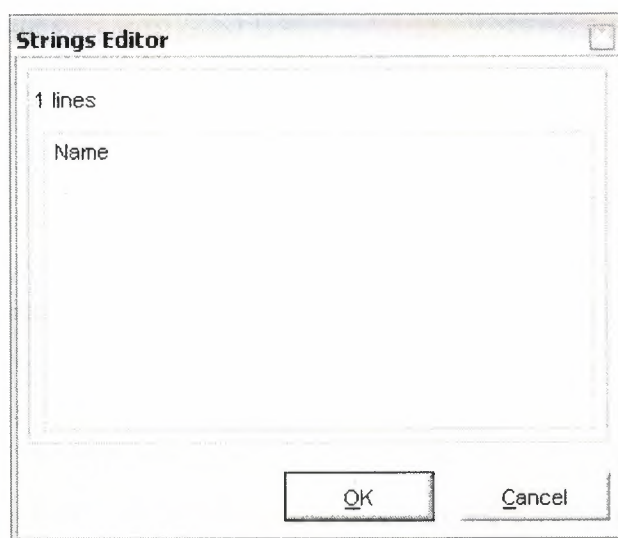
If you can use parameters in your reports. They can be defined using the parameters property of either the Project, a Report or a Page. Parameters can be defined in either of these places, they are just in multiple places for easier access.

You can only select the Project and a Report through the Project Treeview. A page, however, can be selected using the Project Treeview or clicking on it's title above the page designer.

Among other uses, you can print parameters. So, for instance, if the title of your report can be user-defined, you could pass it from your application into the report as a parameter.

Let's add a new report to this project to see how parameters work. To do that, click the fourth button on the toolbar or choose File/New Report. Call it ParametrizedReport, changing its name through the object inspector. This report is going to be very similar to the first one, except the text is going to be user-defined.

Now we need to define the parameter that is going to be printed. To do that, still having the report as the selected object, open the property editor the the parameters property. There should be listed all parameters of this report, each on a separate line. Add a parameter called Name, like this:



**Figure 2.26** Strings Editor

Parameters can be printed using a DataText component, available in the Report tab of the component pallette. Add a DataText to the page, and open the property editor of the DataField property. There you can choose which field is going to be printed, when working with DataAware reports. You can also choose Project Variables, Parameters and Post-Initialize Variables from there.

So choose the parameter added previously from the Parameters drop-down combo and press the Insert Parameter button. The data text expression is now Param.Name. Press OK and try to execute the report, as before. Nothing is printed, since the parameter has not been set.

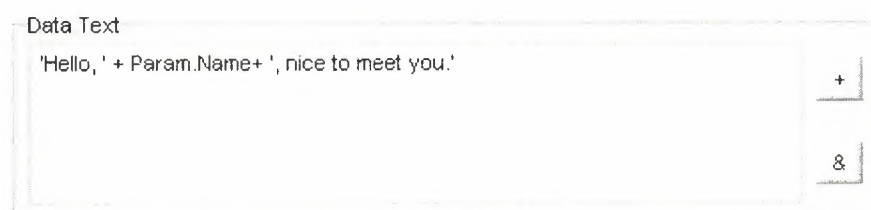
We need to set this parameter before printing. Don't forget to save your changes, and return to Delphi, adding a call to SelectReport before Execute, so we can see the right report. Before executing, though, we need to set the parameter we added. That is made using RvProject's SetParam method. This is how my code looks like right now:

```
procedure TFormMain.btnExecuteClick(Sender: TObject);
begin
    RvProject.Open;
    RvProject.SelectReport('ParametrizedReport',False);
    RvProject.SetParam('Name','Leonel');
    RvProject.Execute;
    RvProject.Close;
end;
```

Now, when we execute the report, we are going to see the string we set as a parameter printed.

Tip: You can use RvProject.GetReportList to get a list of available projects, and add them to a ComboBox, or a RadioGroup, for example. That makes selecting the report easier.

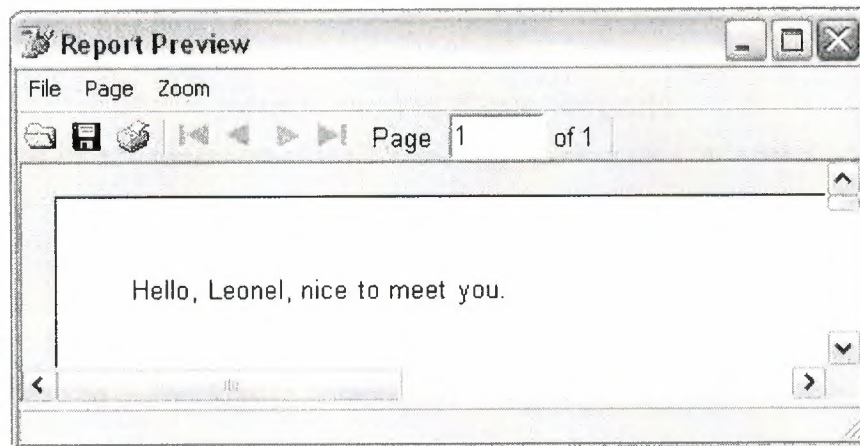
But this is too simple. Let's change the expression that is going to be printed. Return to Rave Designer and open the property editor for the DataText we added. You can add any text you want, combining text, fields, parameters and variables. I changed it to this:



**Figure 2.27** Data Text Sample



Here's the result:



**Figure 2.28** Report Preview

### Post-Initialize Variables

Post-Initialize Variables, or simply PI Vars, are variables whose value is only known after the report has already been printed. It may sound strange, at first, but think about the number of pages of a report, for example. We can only know its value after the report is ready. Actually TotalPages is a report variable that acts like a PI var, and can easily be printed using DataTexts as we did with Parameters.

### Global Pages

When you have parts of reports that are common to two or more reports, you can put these in a global page. Let's suppose we have a header with our company name, the date and time that report is being printed, the current page and the number of pages of that report. We want that header to be in every report. How can we do it?

First, add a global page to the project, using File/New Global Page, or the Toolbar shortcut. In that page, add a section component, available in the standard tab of the component palette.

Sections are logical groupings of components. They can be used to group component so they can be easily moved around the report or as containers for Mirrors, as we are doing right now.

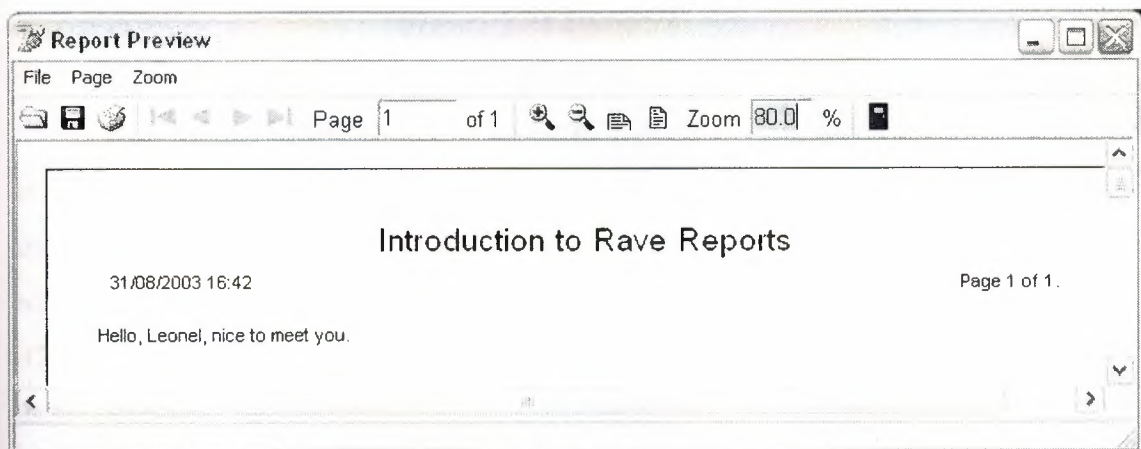
Inside that section we add what we want to be printed. In this case, a few DataTexts. My header looks like this:



**Figure 2.29** Header Sample

Hint: Instead of changing the font property of several components to the same font, link them to a FontMaster component, available in the standard tab, and set the font on it. That way is easier to change the font in the future, in case it's needed.

Now add another section to the Page1 of SimpleReport. Set its Mirror property to GlobalPage1.Section1. You will see a copy of the header you created in the global page. Do the same thing to ParametrizedReport. Now both reports share the same header. Here how it looks like:



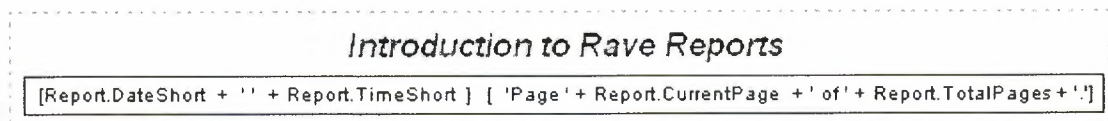
**Figure 2.30** Report Preview

### Conditional Printing

Sometimes we need to print certain parts of a reporting depending of some conditions. Rave has a very powerful way of dealing with this. We can conditionally mirror sections depending on field values or parameters. Let's create a new Report, calling it a ConditionalReport.

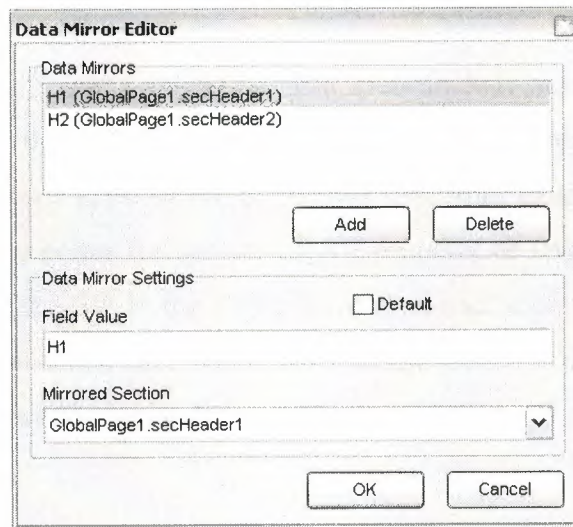
Let's pretend that this new report is a trick one. The user can choose the header that is going to be printed, from two different kinds of headers. He can also choose for the report to be printed without a header. We are going to use a parameter to tell the report what kind of header is going to be printed, and a DataMirrorSection to select the proper header at runtime.

First, add a parameter to this new report called HeaderKind. Let's assume that it will have the values H0 (for no header), H1 (for the first header), H2 (for the second kind of header). Now add a new section to the global page (you can reach it through the Project Treeview), with the second kind of header layout. I created a header similar to the first one, changing the font title and adding a border around the values. It looks like this:



**Figure 2.31** Header Sample

Now return to the Page1 of ConditionalReport, and add a DataMirrorSection, available at the Report tab of the component palette. Go to its DataField property editor, and set Param.HeaderKind as the expression. Now go to the DataMirrors property editor, and add two Data Mirrors: if the value is H1, it should point to the first header, H2, to the second. Since H0 does not match any mirrors, nothing will be printed. It should look like this:



**Figure 2.32** Data Mirror Editor

Notice that I gave more meaningful names to each of the sections earlier.

Hint: You can use the `OnMirrorValue` event of the `DataMirrorSection` to work on ranges of values.

Now return to Delphi and add the code to set the parameter according to the user's choice. I added a `ComboBox` with the options and my code looks like this:

```
Procedure TFormMain.btnExecuteClick(Sender: TObject);
```

```
Begin
```

```
  RvProject.Open;
```

```
  RvProject.SelectReport (cmbReports.Text, False);
```

```
  case cmbReports.ItemIndex of
```

```
    1: RvProject.SetParam('Name',edName.Text);
```

```
    2: RvProject.SetParam('HeaderKind',Format('H%d',[cmbHeaderKind.ItemIndex]));
```

```
  end;
```

```
  RvProject.Execute;
```

```
  RvProject.Close;
```

```
end;
```

Now the proper header will be printed according to the user's choice.

Embedding the Project in the Executable



When you deploy your application, you must include your project file. You can have it as a separated file, so you can update it in a easier way, only shipping a new one, without recompiling your application, or include it in your executable. It's easy to do that: open the property editor for the StoreRAV property of RvProject. There you can press Load to include the file in the DFM, Save to extract a previously saved file, and Clear to remove an embedded file. When there's a file loaded in this property, you don't need to ship the project file separately.

## **2.11 Data Aware Reports**

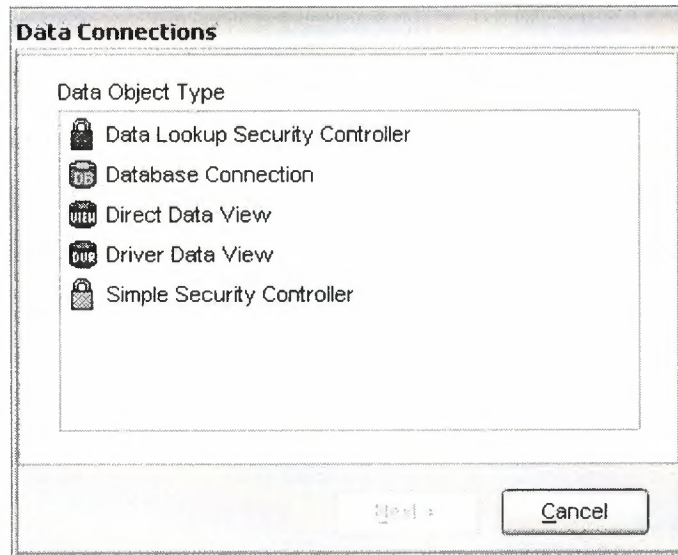
### **2.11.1 The Database Connection**

There are two ways to access data from inside a report: you can share the same connection established by your application, fetching records from Datasets that exists in your Forms or Datamodules, or you can configure a new connection on the report, allowing it to be independent of a particular application. For the first method you would use a Direct Data View and a Driver Data View for the second. Data View is the analog of a DataSource/DataSet combination inside the report.

If you intend to deploy your application using Nevrona's Rave Report Server, you should use Driver Data Views.

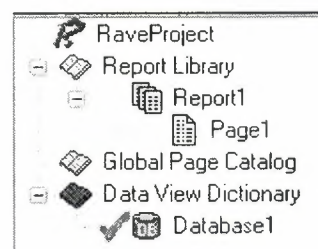
### **2.11.2 The Driver Data View**

Let's create a simple database report using a Driver Data View. Start the Rave Visual Designer, and start a new project. We need to define the database connection. To do this, choose File/New Database Object, or press the sixth button in the toolbar (the purple cube). The Data Connections window will appear:



**Figure 2.33** Data Connection Window

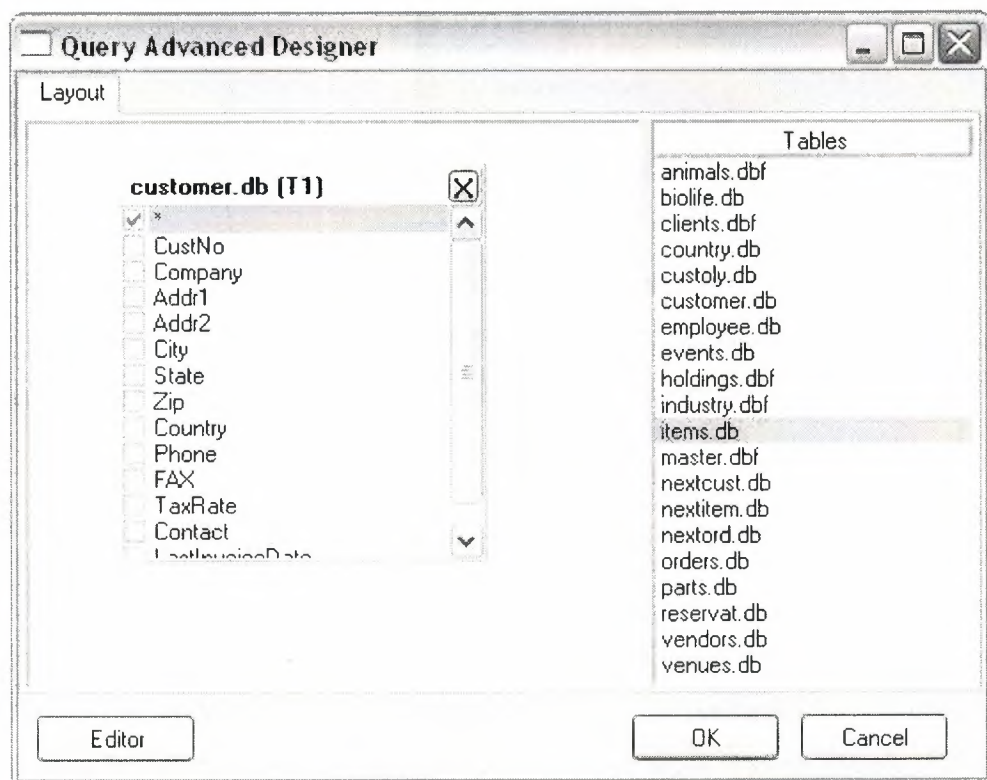
Choose Database Connection, and you will be asked which Data Link you are going to be using. There is a folder called DataLinks where Rave has been installed, containing some files with the .rvd extensions, responsible from the connection mechanism. By default, you can choose between BDE, DbExpress and ADO. I'll be using BDE for this example. Choose BDE; press Finish, and the Database Connection Parameters window will show up. Every Data Link has a different set of connection parameters available, similar to those available in the Delphi IDE. For now, just set Alias to DbDemos and press OK. Notice that a Database object has been added to the Project Treeview, under Data View Dictionary:



**Figure 2.34** Project Tree View

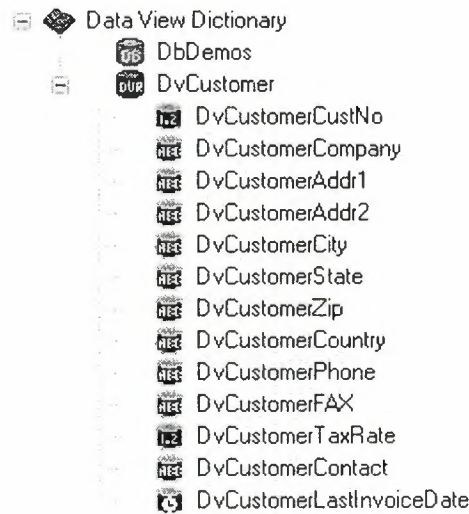
Notice that the settings you configured in the Database Connection Parameters, after the wizard, including username and password, if applicable, were saved in the AuthDesign property of the Database component. In the AuthRun property you can use different settings to be used at runtime, when your report has been deployed.

We are going to create now the Driver Data View. Click on New Data Object, and then choose Driver Data View. You should now choose the Database Connection that is going to be used by this Data View: choose the Database created in the previous step. A Query Advanced Designer will show up. Drag and Drop the table customer.db from the table list to the Layout window. It should look like this:



**Figure 2.35** Query Advanced Designer Window

If you have more than one table, you should drag and drop fields that should be joined between tables. If you press the Editor Button you can check the generated SQL, or type-in a more complex query. Let's keep the simple Customer Listing for now. Press OK and a DriverDataView will be added to the Project Treeview, below the Database components, having the selected fields as subitems:



**Figure 2.36** Project Tree View

Notice that I renamed the Database Connection and the Data View to more appropriate names. It's in the Treeview where properties of the fields should be set, like the Display Label (FullName property), and the DisplayFormat.

### 2.11.3 Regions and Bands

Report components that should be printed in a fixed position in every page, like fixed headers and footers can be put directly in page. Components, whose position will be dependent of previously printed items, should be put in bands. DataBands will be printed once for every record in the linked DataView, while regular Bands will only be printed once, regardless of how many records have been selected. Both can contain Data-Aware components (like DataText), or regular components (like Text).

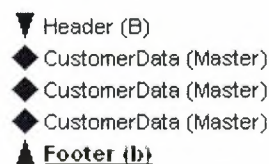
Bands should be put inside Regions. Regions delimitate the width of the bands, and the maximum height that bands can use before starting a new page. One page can have many Regions, and one Region can contain many Bands.

Add a Region to the Page covering its whole area. Inside the region add a Band, to be used as the report header, a DataBand, to print the customer information, and another Band, the report footer.



If you wish to change the ordering of existing bands in a report, use the Move Forward and Move Behind buttons in the Alignment Toolbar.

Rename the bands to more meaningful names (I used Header, CustomerData and Footer). Set the `DataSource` property of CustomerData to `DvCustomer`, and set CustomerData as the `ControllerBand` of the Header and Footer bands. You should also run the Band Style Editor, from the Object Inspector, and set the Print Location of those two bands to Body Header and Body Footer, respectively. You can have an idea on how the report is going to be printed observing the Band Display as you change the settings. It shows iterating bands repeated three times, and other bands only once:

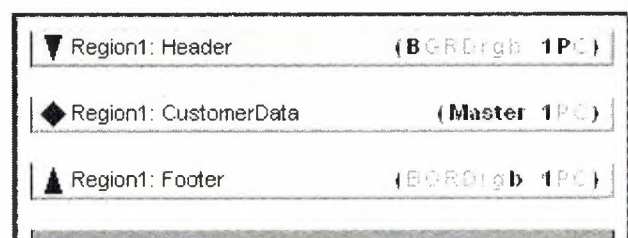


**Figure 2.37** Band Display

We also want the Header to be printed in other pages in case the listing spans more than one page: check the New Page option in the Print Occurrence groupbox, in that same dialog.

The Footer band will only print when `DvCustomers` has reached its end. If you want it printed in every page, regardless of that, just put the components directly on the page, below the region, and not in a Band.

In the editor, you can quickly identify the relationship between bands, their styles and their print occurrences:



**Figure 2.38** Editor Sample

#### **2.11.4 Adding Fields**

It's not hard to add fields to a report. You can Ctrl+Drag the fields from the DataView, in the Project Treeview, to add DataText components to the report, and Alt+Drag them to add Text components containing the Fullname property. This allows you to quickly create the layout of the report. Now add some fields to CustomerData and their title to the Header. I added CustNo, Company, Phone, TaxRate and LastInvoiceDate.

Don't forget that you can use the tools on the Alignment Toolbar to align the components, even if they are in different bands.

I added a title to the Header band and a simple text to the Footer band, indicating that the listing has ended. Later on the series we are going to see how to use the CalcOp and CalcTotal components to be able to add totals, averages and other calculated values to the Footer.

#### **2.11.5 Adding the Report to Your Project**

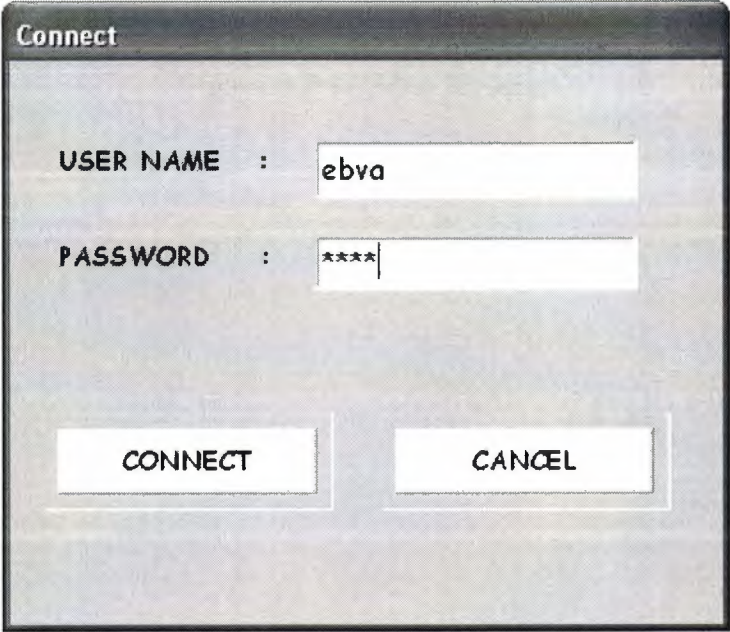
To add this report to your project you should use the same approach as seen in Part II: just use a RvProject in a Form or DataModule, link it to the report file, and call its Execute method. But there is one gotcha when using Driver Data Views: your application must load the appropriate driver. To do that, just add the unit RvDLBDE to your uses clause, if using BDE, RvDLDBX if using DbExpress, or RvDLADO if using ADO.

## CHAPTER THREE

### 3 STOCK PROPERTY BY USING DELPHI

#### 3.1 Database Connection Screen

When user executes program, first database connection screen appears. In this screen user enters user name and password to use the program. so user must have a valid user name and password. Also user must have appropriate privileges on database; such as view, add, update, delete.

A screenshot of a Delphi-style database connection dialog box titled "Connect". It features two input fields: "USER NAME" with the text "ebva" and "PASSWORD" with masked characters "\*\*\*\*". At the bottom are two buttons labeled "CONNECT" and "CANCEL".

Connect

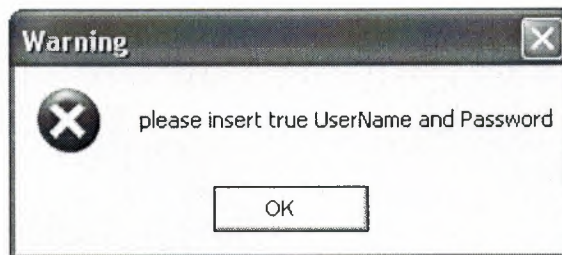
USER NAME : ebva

PASSWORD : \*\*\*\*

CONNECT CANCEL

Figure 3.1 Database Connection Screen

If user name or passwords are not entered correctly a screen appears with a message as "please insert true UserName and Password".



**Figure 3.2** Warning Message



### 3.2 Main Menu

When the user name and password are entered correctly user meets the Main Menu screen. As you can see in this figure there are 15 sections; house to let, house for sale, shop to let, shop for sale, plot for sale, garden for sale, building for sale, farm for sale, villa for sale, field for sale, flier print, about, informations, user register and exit are the names of the sections.



Figure 3.3 Main Menu

### 3.3 House to Let Menu

In house to let menu user can organize, search and print of house to let.

#### 3.3.1 House to let Organize Form

House to let organize form have 8 sections. The sections will be explain below.

New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already rented checkbox signed it means this house already rented otherwise if it is not signed it means it is available for let. Houseowner informations show the information of owner. Buyer informations show the information of customer. House to let informations show the information of house.

The screenshot shows a software window titled "house\_to\_let" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a toolbar at the top with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there are three main sections:

- Search and Action Buttons:** A "SEARCH" button with a magnifying glass icon, a checkbox labeled "Already rented" which is checked, and a "PRINT" button with a printer icon.
- HOUSEOWNER INFORMATIONS:** A section with three text input fields:
  - Name Surname : CEMALİYE DEMİR
  - Cell Phone : 0533 765 34 32
  - Other Phone : 0392 223 12 45
- BUYER INFORMATIONS:** A section with three text input fields:
  - Name Surname : AKAY ÇOLAK
  - Cell Phone : 0 533 456 21 56
  - Other Phone : 0392 453 12 23
- HOUSE TO LET INFORMATIONS:** A section with multiple text input fields arranged in two columns:
  - Registration Date : 12/03/1996
  - Square Meter : 125
  - District : GÖNYELİ
  - Type : STUDYO EV
  - Address : ALİVEHİT SK. LEVENT APT. NO:12 LEFKOŞA/KKTC
  - Price : 70.000,00 TL
  - Aspect : KUZEY
  - Floor : 3
  - Heating System : SOBA

Figure 3.4 House to Let Organize Form



### 3.3.2 House to Let Search Form

House to let search form show to user detailed information and same time you can search the houses available for customer. Using preview button you can go to initial form.

Registrationdate	Squaremeter	District	Type	Price	Condition
12/03/1996	125	GÖNYELİ	STUDYO EV	70.000,00 TL	True
21/04/2006	260	ORTAKÖY	3+1	90.000,00 TL	False

Figure 3.5 House to Let Search Form

At house to let search part you can search available houses for letting according to their features. If the condition of house is false it means the house is empty you can let the house. If the condition is true it means you can not let the house because the house is already was letted.

housetolet\_search

RESEARCH

Search as to Sqare Meter : 260

Search as to District :

Search as to Price :

Search as to Heating System :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition
21/04/2006	260	ORTAKÖY	3+1	90.000,00 TL	False

**Figure 3.6** House to Let Search Form in Edit Mode



If you press previous section you can go to the current page of available house.

house\_to\_let

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ this house give to let PRINT

**HOUSEOWNER INFORMATIONS**

Name Surname : SEDA ÇİÇEK

Cell Phone : 0542 874 24 34

Other Phone : 0392 273 78 67

**HOUSE TO LET INFORMATIONS**

Registration Date : 21/04/2006 Price : 90.000,00 TL

Square Meter : 260 Aspect : KUZHEY

District : ORTAKÖY Floor : 3

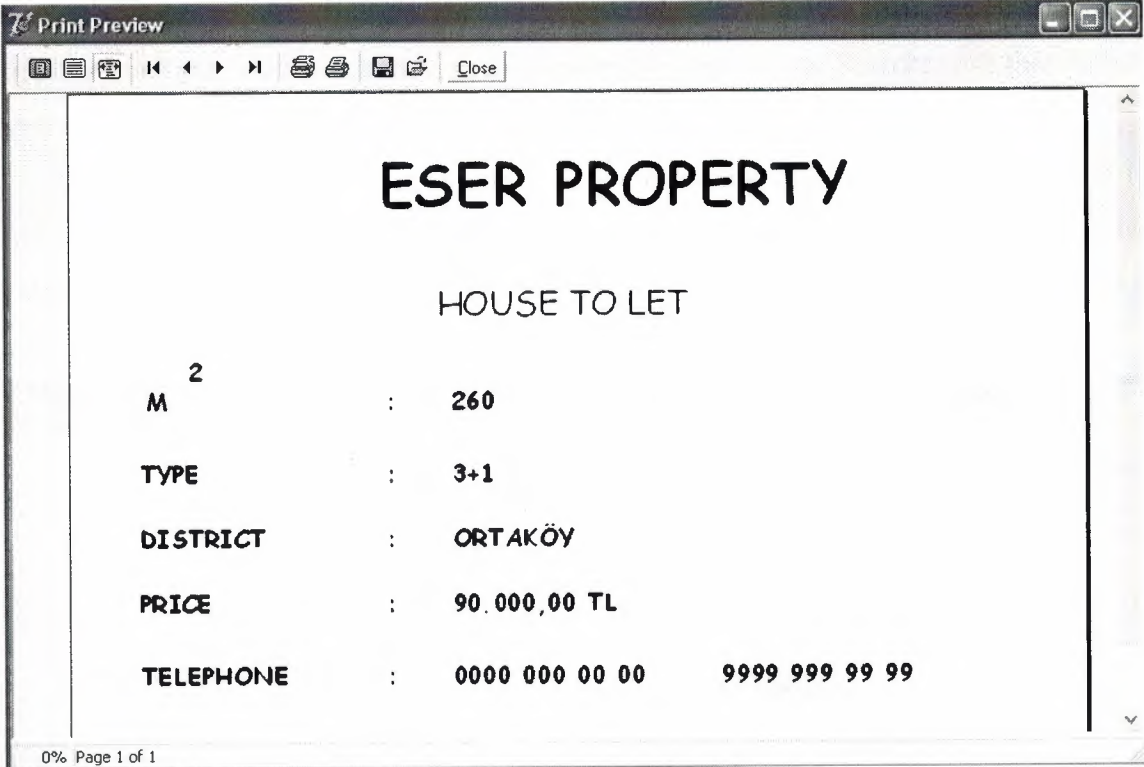
Type : 3+1 Heating System : DOĞALGAZ

Address : HUZURLU SK.SELEN APT.NO:7 LEFKOŞA/KKTC

Figure 3.7 House to Let Organize Form in Edit Mode

### 3.3.3 House to Let Report Form

Using house to let report form you can print the informations about that house.



The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, save, and other functions, along with a 'Close' button. The main content area displays the following information:

**ESER PROPERTY**

HOUSE TO LET

M <sup>2</sup>	:	260
TYPE	:	3+1
DISTRICT	:	ORTAKÖY
PRICE	:	90.000,00 TL
TELEPHONE	:	0000 000 00 00      9999 999 99 99

0% Page 1 of 1

**Figure 3.8** House to Let Report Form



### 3.4 House for Sale Menu

In house for sale menu user can organize, search and print of house for sale.

#### 3.4.1 House for Sale Organize Form

House for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this house already sold otherwise if it is not signed it means it is available for sale. Houseowner informations show the information of owner. Buyer informations show the information of customer. House for sale informations show the information of house.

The screenshot shows a software window titled "house\_for\_sale" with standard Windows window controls. The interface includes a top toolbar with buttons: NEW, PREVIOUS, NEXT, CLEAR, CANCEL, and SAVE. Below the toolbar, there is a SEARCH button with a magnifying glass icon and a PRINT button with a printer icon. A checkbox labeled "Already sold" is checked. The form is divided into three main sections: HOUSEOWNER INFORMATIONS, BUYER INFORMATIONS, and HOUSE FOR SALE INFORMATIONS. Each section contains several text input fields for personal and property details.

HOUSEOWNER INFORMATIONS		BUYER INFORMATIONS	
Name Surname :	AHMET ESER	Name Surname :	EKREM PALTA
Cell Phone :	0532 234 56 98	Cell Phone :	0533 237 45 76
Other Phone :	0392 245 76 43	Other Phone :	0392 654 32 87

HOUSE FOR SALE INFORMATIONS			
Registration Date :	14/08/1990	Price :	85.000,00 TL
Square Meter :	225	Aspect :	BATI
District :	TAŞKINKÖY	Floor :	4
Type :	2+1	Heating System :	DOĞALGAZ
Address :	HANZADE SK.LEVENT APT.NO:12 LEFKOŞA/KKTC		

Figure 3.9 House for Sale Organize Form

### 3.4.2 House for Sale Search Form

House for sale search form show to user detailed information and same time you can search the houses available for customer.Using preview button you can go to initial form.

houseforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	14/08/1990	225	TAŞKINKÖY	2+1	85.000,00 TL	True	
	15/11/1999	145	HAMİTKÖY	3+1	66.000,00 TL	False	

Figure 3.10 House for Sale Search Form



At house for sale search part you can search available houses for selling according to their features. If the condition of house is false it means the house is empty you can sale the house. If the condition is true it means you can not sale the house because the house is already was sold.

houseforsale\_search


RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :


Search as to Heating System :

 Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	
▶	15/11/1999	145	HAMİTKÖY	3+1	66.000,00 TL	False	^

**Figure 3.11** House for Sale Search in Edit Mode

If you press previous section you can go to the current page of available house.



**house\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ Sell house PRINT

**HOUSEOWNER INFORMATION**

Name Surname : MEHMET KAYA

Cell Phone : 0543 234 11 55

Other Phone : 0392 876 54 32

**HOUSE FOR SALE INFORMATION**

Registration Date : 15/11/1999 Price : 66.000,00 TL

Square Meter : 145 Aspect : DOĞU

District : HAMİTKÖY Floor : 2

Type : 3+1 Heating System : SOBA

Address : DEMİRLİ MH. ÇELSA SK.NO:23 LEFKOŞA/KKTC

**Figure 3.12** House for Sale Organize Form in Edit Mode

### 3.4.3 House for Sale Report Form

Using house to let report form you can print the informations about that house.

**ESER PROPERTY**

**HOUSE FOR SALE**

M <sup>2</sup>	:	145	
TYPE	:	3+1	
DISTRICT	:	HAMİTKÖY	
PRICE	:	66.000,00 TL	
TELEPHONE	:	0532 345 21 34	0542 843 77 59

0% Page 1 of 1

**Figure 3.13** House for Sale Report



### 3.5 Shop to Let Menu

In shop to let menu user can organize, search and print of shop to let.

#### 3.5.1 Shop to Let Organize Form

Shop to let organize form have 8 sections. The sections will be explain below.

New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already rented checkbox signed it means this shop already rented otherwise if it is not signed it means it is available for let. Owner of a shop informations show the information of owner. Buyer informations show the information of customer. Shop to let informations show the information of shop.

The screenshot shows a software window titled "shop\_to\_let" with standard Windows window controls (minimize, maximize, close) in the top right corner. Below the title bar is a horizontal toolbar with six buttons: "NEW" (with a document icon), "PREVIOUS" (with a left arrow icon), "NEXT" (with a right arrow icon), "CLEAR" (with an 'x' icon), "CANCEL" (with a trash can icon), and "SAVE" (with a floppy disk icon). Below the toolbar, there is a "SEARCH" button with a magnifying glass icon, a checkbox labeled "Already rented" which is checked, and a "PRINT" button with a printer icon. The main area of the form is divided into three sections. The first section, "OWNER OF A SHOP INFORMATIONS", contains three text input fields: "Name Surname" with the value "TAHIR ÖZDEMİR", "Cell Phone" with the value "0542 865 45 67", and "Other Phone" with the value "0392 865 34 54". The second section, "BUYER INFORMATIONS", also contains three text input fields: "Name Surname" with the value "EVREN ÇAMLI", "Cell Phone" with the value "0533 764 89 21", and "Other Phone" with the value "0392 751 39 30". The third section, "SHOP TO LET INFORMATIONS", contains several text input fields arranged in two columns. The left column includes "Registration Date" (03/11/2003), "Square Meter" (55), "District" (DEREBOYU), "Type" (PASAJ), and "Address" (SARMAŞIK SK.NO:5 LEFKOŞA/KKTC). The right column includes "Price" (45.000,00 TL), "Aspect" (BATI), "Floor" (1), and "Heating System" (DOĞALGAZ).

Figure 3.14 Shop to Let Organize Form



### 3.5.2 Shop to Let Search Form

Shop to let search form show to user detailed information and same time you can search the shops available for customer.Using preview button you can go to initial form.

**shoptolet\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	03/11/2003		55 DEREBOYU	PASAJ	45.000,00 TL	True	
□	30/01/1989		85 YENİKENT	GALERİ	45.000,00 TL	False	

Figure 3.15 Shop to Let Search Form

At shop to let search part you can search available shops for letting according to their features. If the condition of shop is false it means the shop is empty you can let the shop. If the condition is true it means you can not sale the shop because the shop is already was letted.

shoptolet\_search

RESEARCH

Search as to Square Meter : 85

Search as to District :

Search as to Price :

Search as to Heating System :


Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	
▶	30/01/1989	85	YENIKENT	GALERİ	45.000,00 TL	False	⌵

**Figure 3.16** Shop to Let Search Form in Edit Mode



If you press previous section you can go to the current page of available shop.



**shop\_to\_let**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ This shop give to let PRINT

**OWNER OF A SHOP INFORMATIONS**

Name Surname : CEM DÜNDAR

Cell Phone : 0533 245 67 87

Other Phone : 0392 754 34 90

**SHOP TO LET INFORMATIONS**

Registration Date : 30/01/1989 Price : 45.000,00 TL

Square Meter : 85 Aspect : GÜNEY

District : YENIKENT Floor : 1

Type : GALERİ Heating System : SOBA

Address : HELEGAN SK. NO:4 LEFKOŞA/KKTC

Figure 3.17 Shop to Let Organize Form in Edit Mode

### 3.5.3 Shop to Let Report Form

Using shop to let report form you can print the informations about that shops.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, copy, paste, and other functions. The main content area displays the following information:

**ESER PROPERTY**

**SHOP TO LET**

M <sup>2</sup>	:	85
TYPE	:	GALERİ
DISTRICT	:	YENİKENT
PRICE	:	45.000,00 TL
TELEPHONE	:	0532 345 21 34      0542 843 77 59

0% Page 1 of 1

Figure 3.18 Shop to Let Report



### 3.6.2 Shop for Sale Search Form

Shop for sale search form show to user detailed information and same time you can search the shops available for customer.Using preview button you can go to initial form.

shopforsale\_search


**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	18/06/2002		55 GİRNEKAPI	ZEMİN	38.000,00 TL	True	
□	23/12/1996		75 KÜÇÜKKAYMAKL	ZEMİN	65.000,00 TL	False	

Figure 3.20 Shop for Sale Search Form

At shop for sale search part you can search available shops for selling according to their features.If the condition of shop is false it means the shop is empty you can sale the shop.If the condition is true it means you can not sale the shop because the shop is already was sold.

shopforsale\_search


RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System : DOĞALGAZ

 preview

Registrationdate	Squaremeter	District	Type	Price	Condition	
18/06/2002	55	GİRNEKAPI	ZEMİN	38.000,00 TL	True	

**Figure 3.21** Shop for Sale Search Form in Edit Mode



### 3.6 Shop for Sale Menu

In shop for sale menu user can organize, search and print of shop for sale.

#### 3.6.1 Shop for Sale Organize Form

Shop for sale organizes form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this shop already sold otherwise if it is not signed it means it is available for sale. Owner of a shop informations show the information of owner. Buyer informations show the information of customer. Shop for sale informations show the information of shop.

The screenshot shows a Windows-style application window titled "shop\_for\_sale". At the top, there is a toolbar with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there is a "SEARCH" button with a magnifying glass icon and a checkbox labeled "Already sold" which is checked. To the right of the checkbox is a "PRINT" button with a printer icon. The main area of the form is divided into three sections: "SHOPKEEPER INFORMATIONS", "BUYER INFORMATIONS", and "SHOP FOR SALE INFORMATIONS". Each section contains several text input fields for personal and property details.

SHOPKEEPER INFORMATIONS		BUYER INFORMATIONS	
Name Surname :	HÜSEYİN EREN	Name Surname :	YASİN KARLI
Cell Phone :	0543 285 23 18	Cell Phone :	0533 284 54 69
Other Phone :	0392 854 23 45	Other Phone :	0392 843 12 65

SHOP FOR SALE INFORMATIONS			
Registration Date :	18/06/2002	Price :	38.000,00 TL
Square Meter :	55	Aspect :	BATI
District :	GİRNEKAPI	Floor :	1
Type :	ZEMİN	Heating System :	DOĞALGAZ
Address :	YARALI SK.NO:9 LEFKOŞA/KKTC		

Figure 3.19 Shop for Sale Organize Form

If you press previous section you can go to the current page of available shop.

shop\_for\_sale

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☒ Already sold PRINT

**SHOPKEEPER INFORMATIONS**

Name Surname : HÜSEYİN EREN

Cell Phone : 0543 285 23 18

Other Phone : 0392 854 23 45

**BUYER INFORMATIONS**

Name Surname : YASİN KARLI

Cell Phone : 0533 284 54 69

Other Phone : 0392 843 12 65

**SHOP FOR SALE INFORMATIONS**

Registration Date : 18/06/2002 Price : 38.000,00 TL

Square Meter : 55 Aspect : BATI

District : GİRNEKAPI Floor : 1

Type : ZEMİN Heating System : DOĞALGAZ

Address : YARALI SK.NO:9 LEFKOŞA/KKTC

Figure 3.22 Shop for Sale Organize Form in Edit Mode



### 3.6.3 Shop for Sale Report Form

Using shop for sale report form you can print the informations about that shops.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, copy, paste, and other functions. The main content area displays a report form for 'ESER PROPERTY'. The form includes the following details:

<b>ESER PROPERTY</b>		
<b>SHOP FOR SALE</b>		
M <sup>2</sup>	:	75
TYPE	:	ZEMİN
DISTRICT	:	KÜÇÜKKAYMAKLI
PRICE	:	65.000,00 TL
TELEPHONE	:	0532 345 21 34      0542 843 77 59

At the bottom of the window, it says '0% Page 1 of 1'.

**Figure 3.23** Shop for Sale Report Form

### 3.7 Plot for Sale Menu

In plot for sale menu user can organize, search and print of plot for sale.

#### 3.7.1 Plot for Sale Organize Form

Plot for sale organizes form have 8 sections. The sections will be explain below.  
New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this plot already sold otherwise if it is not signed it means it is available for sale. Owner of a plot informations show the information of owner. Buyer informations show the information of customer. Plot for sale informations show the information of plot.

The screenshot shows a software window titled "plot\_for\_sale". At the top, there is a toolbar with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there is a "SEARCH" button with a magnifying glass icon and a "PRINT" button with a printer icon. In the center, there is a checkbox labeled "Already sold" which is checked. Below this, the form is divided into three main sections: "OWNER OF A PLOT INFORMATIONS", "BUYER INFORMATIONS", and "PLOT FOR SALE INFORMATIONS". The "OWNER OF A PLOT INFORMATIONS" section contains three input fields: "Name Surname" (MURAT KARAHAN), "Cell Phone" (0542 645 39 65), and "Other Phone" (0392 194 28 79). The "BUYER INFORMATIONS" section contains three input fields: "Name Surname" (MAHIR SEREN), "Cell Phone" (0533 943 29 54), and "Other Phone" (0392 943 29 21). The "PLOT FOR SALE INFORMATIONS" section contains five input fields: "Registration Date" (27/10/1986), "District" (SEFAKÖY), "Square Meter" (320), "Price" (85.000,00 TL), and "Type" (YOLÜSTÜ). At the bottom, there is an "Address" field with the text "HASTANE KARŞISI HAŞMET SK. LEFKOŞA/KKTC".

OWNER OF A PLOT INFORMATIONS	
Name Surname :	MURAT KARAHAN
Cell Phone :	0542 645 39 65
Other Phone :	0392 194 28 79

BUYER INFORMATIONS	
Name Surname :	MAHIR SEREN
Cell Phone :	0533 943 29 54
Other Phone :	0392 943 29 21

PLOT FOR SALE INFORMATIONS			
Registration Date :	27/10/1986	District :	SEFAKÖY
Square Meter :	320	Price :	85.000,00 TL
Type :	YOLÜSTÜ		
Address :	HASTANE KARŞISI HAŞMET SK. LEFKOŞA/KKTC		

Figure 3.24 Plot for Sale Organize Form

### 3.7.2 Plot for Sale Search Form

Plot for sale search form show to user detailed information and same time you can search the plots available for customer.Using preview button you can go to initial form.


**plotforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

 Preview

Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶ 27/10/1986	320	SEFAKÖY	YOLÜSTÜ	85.000,00 TL	True	
15/02/2000	400	LEMAR	PARKYANI	95.000,00 TL	False	

Figure 3.25 Plot for Sale Search Form



At plot for sale search part you can search available plots for selling according to their features.If the condition of plot is false it means the plot is empty you can sale the plot.If the condition is true it means you can not sale the plot because the plot is already was sold.

plotforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

 Preview

Registrationdate	Squaremeter	District	Type	Price	Condition	
▶ 15/02/2000	400	LEMAR	PARKYANI	95.000,00 TL	False	⬆

Figure 3.26 Plot for Sale Search Form in Edit Mode



If you press previous section you can go to the current page of available plot.



**plot\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ Sell plot PRINT

**OWNER OF A PLOT INFORMATIONS**

Name Surname : NURİ ERTAŞ

Cell Phone : 0533 296 33 65

Other Phone : 0392 743 21 84

**PLOT FOR SALE INFORMATIONS**

Registration Date : 15/02/2000 District : LEMAR

Square Meter : 400 Price : 95.000,00 TL

Type : PARKYANI

Address : FENERLİ SK. İLKÖĞRETİM KARŞISI LEFKOŞA/KKTC

**Figure 3.27** Plot for Sale Organize Form in Edit Mode

### 3.7.3 Plot for Sale Report Form

Using plot for sale report form you can print the informations about that plots.

The screenshot shows a 'Print Preview' window with a toolbar containing icons for print, copy, paste, and navigation. The main content area displays the following information:

**ESER PROPERTY**

**PLOT FOR SALE**

M <sup>2</sup>	:	320	
TYPE	:	YOLÜSTÜ	
DISTRICT	:	SEFAKÖY	
PRICE	:	85.000,00 TL	
TELEPHONE	:	0532 345 21 34	0542 843 77 59

0% Page 1 of 1

**Figure 3.28** Plot for Sale Report Form



### 3.8 Garden for Sale Menu

In garden for sale menu user can organize, search and print of garden for sale.

#### 3.8.1 Garden for Sale Organize Form

Garden for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this garden already sold otherwise if it is not signed it means it is available for sale. Owner of a garden informations show the information of owner. Buyer informations show the information of customer. Garden for sale informations show the information of garden.

The screenshot shows a software window titled "garden\_for\_sale". At the top, there is a toolbar with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there are two buttons: "SEARCH" (with a magnifying glass icon) and "PRINT" (with a printer icon). In the center, there is a checkbox labeled "Already sold" which is checked. Below this, the form is divided into three main sections. The first section, "OWNER OF A GARDEN INFORMATIONS", contains three input fields: "Name Surname" with the value "SEFA ATACAN", "Cell Phone" with the value "0533 645 28 56", and "Other Phone" with the value "0392 754 38 59". The second section, "BUYER INFORMATIONS", also contains three input fields: "Name Surname" with the value "FATİH METE", "Cell Phone" with the value "0542 458 84 68", and "Other Phone" with the value "0392 734 29 59". The third section, "GARDEN FOR SALE INFORMATIONS", contains five input fields: "Registration Date" with the value "21/03/1990", "District" with the value "GÖNYELİ", "Square Meter" with the value "220", "Price" with the value "92.000,00 TL", and "Type" with the value "SERA". At the bottom, there is a single input field for "Address" with the value "SEÇKİN SK. AKBANK KARŞISI LEFKOŞA/KKTC".

Figure 3.29 Garden for Sale Organize Form

### 3.8.2 Garden for Sale Search Form

Garden for sale search form show to user detailed information and same time you can search the gardens available for customer.Using preview button you can go to initial form.


**gardenforsale\_search**

**RESEARCH**

Search as to Sqare Meter :

Search as to District :

Search as to Price :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition ^
▶	21/03/1990	220	GÖNYELİ	SERA	92.000,00 TL	True
□	11/09/1992	350	KAYALI	410 KAYISI	88.000,00 TL	False

**Figure 3.30** Garden for Sale Search Form



At garden for sale search part you can search available gardens for selling according to their features. If the condition of garden is false it means the garden is empty you can sale the garden. If the condition is true it means you can not sale the garden because the garden is already was sold.

gardenforsale\_search

RESEARCH

Search as to Sqare Meter : 350

Search as to District :

Search as to Price :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition
11/09/1992	350	KAYALI	410 KAYISI	88.000,00 TL	False

**Figure 3.31** Garden for Sale Search Form in Edit Mode

If you press previous section you can go to the current page of available garden.



**garden\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH PRINT

☐ Sell garden

**OWNER OF A GARDEN INFORMATIONS**

Name Surname : DENIZ SAKAR

Cell Phone : 0542 743 94 98

Other Phone : 0392 732 18 39

**GARDEN FOR SALE INFORMATIONS**

Registration Date : 11/09/1992 District : KAYALI

Square Meter : 350 Price : 88.000,00 TL

Type : 410 KAYISI

Address : GÜZELYURT YOLU ÇAMLIÇA SK. LEFKOŞA/KKTC

**Figure 3.32** Garden for Sale Organize Form in Edit Mode

### 3.8.3 Garden for Sale Report Form

Using garden for sale report form you can print the informations about that gardens.

**ESER PROPERTY**

**GARDEN FOR SALE**

M <sup>2</sup>	:	350	
TYPE	:	410 KAYISI	
DISTRICT	:	KAYALI	
PRICE	:	88.000,00 TL	
TELEPHONE	:	0532 345 21 34	0542 843 77 59

0% Page 1 of 1

**Figure 3.33** Garden for Sale Report Form



### 3.9 Building For Sale Menu

In building for sale menu user can organize, search and print of building for sale.

#### 3.9.1 Building for Sale Organize Form

Building for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this building already sold otherwise if it is not signed it means it is available for sale. Owner of a building informations show the information of owner. Buyer informations show the information of customer. Building for sale informations show the information of building.

The screenshot shows a software window titled "building\_for\_sale" with standard Windows window controls. The interface includes a top toolbar with buttons: NEW, PREVIOUS, NEXT, CLEAR, CANCEL, and SAVE. Below the toolbar, there is a SEARCH button with a magnifying glass icon, a checkbox labeled "Already sold" which is checked, and a PRINT button with a printer icon. The main area is divided into three sections: "OWNER OF A BUILDING INFORMATIONS" with fields for Name Surname (METİN ESENGÜL), Cell Phone (0542 456 67 85), and Other Phone (0392 553 68 49); "BUYER INFORMATIONS" with fields for Name Surname (MAHMUT YİĞİT), Cell Phone (0542 754 39 98), and Other Phone (0392 943 18 45); and "BUILDING FOR SALE INFORMATIONS" with fields for Registration Date (18/08/1995), Price (75.000,00 TL), Square meter (320), Aspect (BATI), District (METROPOL), Floor (5), Type (4+1), Heating System (SOBA), and address (FAZİLET MH.SEREN SK.NO:7 LEFKOŞA/KKTC).

Figure 3.34 Building for Sale Organize Form

### 3.9.2 Building for Sale Search Form

Building for sale search form show to user detailed information and same time you can search the buildings available for customer.Using preview button you can go to initial form.

**buildingforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	18/08/1995	320	METROPOL	4+1	75.000,00 TL	True	
□	01/01/1993	185	BAHÇELİEVLI	3+1	98.765,00 TL	False	

**Figure 3.35** Building for Sale Search Form



At building for sale search part you can search available buildings for selling according to their features. If the condition of building is false it means the building is empty you can sale the building. If the condition is true it means you can not sale the building because the building is already was sold.

buildingforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

75000

Search as to Heating System :

Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	
▶	18/08/1995	320	METROPOL	4+1	75.000,00 TL	True	^

**Figure 3.36** Building for Sale Search Form in Edit Mode



If you press previous section you can go to the current page of available building.

building\_for\_sale

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☒ Already sold PRINT

**OWNER OF A BUILDING INFORMATIONS**

Name Surname : METİN ESENGÜL

Cell Phone : 0542 456 67 85

Other Phone : 0392 553 68 49

**BUYER INFORMATIONS**

Name Surname : MAHMUT YİĞİT

Cell Phone : 0542 754 39 98

Other Phone : 0392 943 18 45

**BUILDING FOR SALE INFORMATIONS**

Registration Date : 18/08/1995 Price : 75.000,00 TL

Square meter : 320 Aspect : BATI

District : METROPOL Floor : 5

Type : 4+1 Heating System : SOBA

address : FAZİLET MH.SEREN SK.NO:7 LEFKOŞA/KKTC

Figure 3.37 Building for Sale Organize Form in Edit Mode

### 3.9.3 Building for Sale Report Form

Using building for sale report form you can print the informations about that buildings.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, zoom, and other functions, along with a 'Close' button. The main content area displays the following information:

**ESER PROPERTY**  
**BUILDING FOR SALE**

M <sup>2</sup>	:	320	
TYPE	:	4+1	
DISTRICT	:	METROPOL	
PRICE	:	75.000,00 TL	
TELEPHONE	:	0532 345 21 34	0542 843 77 59

0% Page 1 of 1

**Figure 3.38** Building for Sale Report Form



### 3.10 Farm for Sale Menu

In farm for sale menu user can organize, search and print of farm for sale.

#### 3.10.1 Farm for Sale Organize Form

Farm for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this farm already sold otherwise if it is not signed it means it is available for sale. Owner of a farm informations show the information of owner. Buyer informations show the information of customer. Farm for sale informations show the information of farm.

The screenshot shows a software window titled "farm\_for\_sale" with standard Windows window controls (minimize, maximize, close). The interface is organized into several sections:

- Navigation Bar:** A row of buttons at the top: NEW, PREVIOUS, NEXT, CLEAR, CANCEL, and SAVE. Each button has a small icon above its text.
- Search and Print Section:** Below the navigation bar, there is a "SEARCH" button with a magnifying glass icon, a checkbox labeled "Already sold" which is checked, and a "PRINT" button with a printer icon.
- OWNER OF A FARM INFORMATIONS:** A section containing three input fields:
  - Name Surname : SONER ERTEGÜL
  - Cell Phone : 0533 574 29 59
  - Other Phone : 0392 483 20 82
- BUYER INFORMATIONS:** A section containing three input fields:
  - Name Surname : CIHAD SELVI
  - Cell Phone : 0533 274 48 28
  - Other Phone : 0392941 26 54
- FARM FOR SALE INFORMATIONS:** A section containing several input fields:
  - Registration Date : 13/12/1991
  - Square Meter : 3200
  - Type : BÜYÜKBAŞ
  - Address : CANDEMİR YOLU 8.KM'DE LEFKOŞA/KKTC
  - District : HAMİTKÖY
  - Price : 87.000,00 TL

Figure 3.39 Farm for Sale Organize Form



### 3.10.2 Farm for Sale Search Form

Farm for sale search form show to user detailed information and same time you can search the farms available for customer.Using preview button you can go to initial form.


**farmforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	13/12/1991	3200	HAMİTKÖY	BÜYÜKBAŞ	87.000,00 TL	True	
▶	29/03/1983	4000	DEMİRHAN	TAVUK	65.000,00 TL	False	

Figure 3.40 Farm for Sale Search Form





If you press previous section you can go to the current page of available farm.



**farm\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ sell farm PRINT

**OWNER OF A FARM INFORMATIONS**

Name Surname : AHMET SEYFI

Cell Phone : 0533 481 39 74

Other Phone : 0392 932 18 47

**FARM FOR SALE INFORMATIONS**

Registration Date : 29/03/1983 District : DEMIRHAN

Square Meter : 4000 Price : 65.000,00 TL

Type : TAVUK

Address : SANAYI YOLU 3.KM'DE LEFKOŞA/KKTC

Figure 3.42 Farm for Sale Organize Form in Edit Mode



### 3.10.3 Farm for Sale Report Form

Using farm for sale report form you can print the informations about that farms.

**Print Preview**

Save Report

**ESER PROPERTY**

**FARM FOR SALE**

2  
M : 4000

TYPE : TAVUK

DISTRICT : DEMIRHAN

PRICE : 65.000,00 TL

TELEPHONE : 0532 345 21 34 0542 843 77 59

0% Page 1 of 1

**Figure 3.43** Farm for Sale Report Form

### 3.11 Villa for Sale Menu

In villa for sale menu user can organize, search and print of villa for sale.

#### 3.11.1 Villa for Sale Organize Form

Villa for sale organize form have 8 sections. The sections will be explain below.  
New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this villa already sold otherwise if it is not signed it means it is available for sale. Owner of a villa informations show the information of owner. Buyer informations show the information of customer. Villa for sale informations show the information of villa.

The screenshot shows a software window titled "villa\_for\_sale" with standard Windows window controls. The interface is organized into several sections:

- Navigation Buttons:** A row of six buttons at the top: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Each button has a small icon above its text.
- Search and Print Section:** Below the navigation buttons, there is a "SEARCH" button with a magnifying glass icon, a checkbox labeled "Already sold" which is checked, and a "PRINT" button with a printer icon.
- OWNER OF A VILLA INFORMATIONS:** A section on the left containing three text input fields:
  - Name Surname : SEDAT YAREN
  - Cell Phone : 0542 374 58 32
  - Other Phone : 0392 963 87 21
- BUYER INFORMATIONS:** A section on the right containing three text input fields:
  - Name Surname : ALI YILDIZ
  - Cell Phone : 0533 852 37 19
  - Other Phone : 0392 563 95 64
- VILLA FOR SALE INFORMATIONS:** A section at the bottom containing several text input fields:
  - Registration Date : 01/09/1998
  - District : ORTAKÖY
  - Square Meter : 320
  - Price : 92.000,00 TL
  - Type : DUBLEX
  - Address : SEFALI MH. SIHIRLI SK.NO:5 LEFKOŞA/KKTC

Figure 3.44 Villa for Sale Organize Form




### 3.11.2 Villa for Sale Search Form

Villa for sale search form show to user detailed information and same time you can search the villas available for customer. Using preview button you can go to initial form.

**villaforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :   **Preview**

Search as to Price :

Registrationdate	Squaremeter	District	Type	Price	Condition	
▶ 01/09/1998	320	ORTAKÖY	DUBLEX	92.000,00 TL	True	^
17/04/2001	225	ERYAMAN	DUBLEX	99.000,00 TL	False	

**Figure 3.45** Villa for Sale Search Form



At villa for sale search part you can search available villas for selling according to their features. If the condition of villa is false it means the villa is empty you can sale the villa.If the condition is true it means you can not sale the villa because the villa is already was sold.

villaforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District : ORTAKÖY

Search as to Price :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition
01/09/1998	320	ORTAKÖY	DUBLEX	92.000,00 TL	True

**Figure 3.46** Villa for Sale Search Form in Edit Mode

If you press previous section you can go to the current page of available villa.

**villa\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☒ Already sold PRINT

**OWNER OF A VILLA INFORMATIONS**

Name Surname : SEDAT YAREN

Cell Phone : 0542 374 58 32

Other Phone : 0392 963 87 21

**BUYER INFORMATIONS**

Name Surname : ALI YILDIZ

Cell Phone : 0533 852 37 19

Other Phone : 0392 563 95 64

**VILLA FOR SALE INFORMATIONS**

Registration Date : 01/09/1998 District : ORTAKÖY

Square Meter : 320 Price : 92.000,00 TL

Type : DUBLEX

Address : SEFALI MH. SİHIRLİ SK.NO:5 LEFKOŞA/KKTC

Figure 3.47 Villa for Sale Organize Form in Edit Mode

### 3.11.3 Villa for Sale Report Form

Using villa for sale report form you can print the informations about that villas.

**Print Preview**

Print Preview window showing the Villa for Sale report form. The form displays the following information:

**ESER PROPERTY**

**VILLA FOR SALE**

**M<sup>2</sup>** : 320

**TYPE** : DUBLEX

**DISTRICT** : ORTAKÖY

**PRICE** : 92.000,00 TL

**TELEPHONE** : 0532 345 21 34      0542 843 77 59

0% Page 1 of 1



## 3.12 Field for Sale Menu

In field for sale menu user can organize, search and print of field for sale.

### 3.12.1 Field for Sale Organize Form

Field for sale organize form have 8 sections. The sections will be explain below.  
New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this field already sold otherwise if it is not signed it means it is available for sale. Owner of a field informations show the information of owner. Buyer informations show the information of customer. Field for sale informations show the information of field.

The screenshot shows a software window titled "field\_for\_sale" with standard Windows window controls (minimize, maximize, close). The window contains several sections and buttons:

- Buttons:** NEW, PREVIOUS, NEXT, CLEAR, CANCEL, SAVE, SEARCH, and PRINT.
- Checkbox:** ☒ Already sold
- OWNER OF A FIELD INFORMATIONS:**
  - Name Surname : CANER TOPBAŞ
  - Cell Phone : 0392 443 51 94
  - Other Phone : 0392 443 51 94
- BUYER INFORMATIONS:**
  - Name Surname : EMIRHAN CANPOLAT
  - Cell Phone : 0533 842 21 56
  - Other Phone : 0392 932 17 43
- FIELD FOR SALE INFORMATIONS:**
  - Registration Date : 19/03/1991
  - District : KANLIDERE
  - Square Meter : 3400
  - Price : 86.000,00 TL
  - Type : 217 ELMA
  - Address : SANDIKLI YOLU 3.KM'DE LEFKOŞA/KKTC

Figure 3.48 Field for Sale Organize Form

### 3.12.2 Field for Sale Search Form

Field for sale search form show to user detailed information and same time you can search the fields available for customer.Using preview button you can go to initial form.


**fieldforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition	
<input checked="" type="checkbox"/>	19/03/1991	3400	KANLIDERE	217 ELMA	86.000,00 TL	True	
<input type="checkbox"/>	21/06/1990	4600	TAŞLIKÖY	KUYUSUZ	75.000,00 TL	False	
<input type="checkbox"/>	21/05/1989	4600	TAŞLIKÖY	105 KAYISI	78.000,00 TL	False	
<input type="checkbox"/>	05/11/2004	2500	DEĞİRMELİK	KUYULU	75.000,00 TL	False	

**Figure 3.49** Field for Sale Search Form



At field for sale search part you can search available fields for selling according to their features. If the condition of field is false it means the field is empty you can sale the field. If the condition is true it means you can not sale the field because the field is already was sold.

fieldforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District : DEĞİRMENLİK

Search as to Price :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition	
05/11/2004	2500	DEĞİRMENLİK	KUYULU	75.000,00 TL	False	

**Figure 3.50** Field for Sale Search Form in Edit Mode



If you press previous section you can go to the current page of available field.

field\_for\_sale

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ Sell field PRINT

**OWNER OF A FIELD INFORMATIONS**

Name Surname : DURMUŞ TEZCAN

Cell Phone : 0392 483 45 68

Other Phone : 0392 483 45 68

**FIELD FOR SALE INFORMATIONS**

Registration Date : 05/11/2004 District : DEĞİRMENLİK

Square Meter : 2500 Price : 75.000,00 TL

Type : KUYULU

Address : HAVAALANI YOLU 8.KM'DE LEFKOŞA/KKTC

Figure 3.51 Field for Sale Organize Form in Edit Mode

### 3.12.3 Field for Sale Report Form

Using field for sale report form you can print the informations about that fields.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, zoom, and other functions, along with a 'Close' button. The main content area displays a report form for 'ESER PROPERTY'. The form includes the following details:

<b>ESER PROPERTY</b>		
<b>FIELD FOR SALE</b>		
2		
M	:	2500
TYPE	:	KUYULU
DISTRICT	:	DEĞİRMENLİK
PRICE	:	75.000,00 TL
TELEPHONE	:	0532 345 21 34      0542 843 77 59

At the bottom of the window, it says '0% Page 1 of 1'.

**Figure 3.52** Field for Sale Report Form



### 3.13 Flier Print Menu

In flier print menu user can print all advertisements.

#### 3.13.1 Flier Print Organize Form

In flier print organize form user can print all advertisements related to each type houses, shops, villas, plots, fields, gardens, buildings and farms.

Using preview button you can go to main menu.

flier\_print

# ESER PROPERTY

 HOUSE TO LET ADVERTISEMENTS	 VILLA FOR SALE ADVERTISEMENTS
 SHOP TO LET ADVERTISEMENTS	 PLOT FOR SALE ADVERTISEMENTS
 HOUSE FOR SALE ADVERTISEMENTS	 FIELD FOR SALE ADVERTISEMENTS
 SHOP FOR SALE ADVERTISEMENTS	 GARDEN FOR SALE ADVERTISEMENTS
 BUILDING FOR SALE ADVERTISEMENTS	 FARM FOR SALE ADVERTISEMENTS

 PREVIEW

Figure 3.53 Flier Print Form



### 3.13.2 House to Let Advertisements Form

Using this form you can print the advertisements about that house to let.



**Print Preview**

**ESER PROPERTY**

**HOUSES TO LET**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
125	GÖNYELİ	STÜDYO EV	70.000,00 TL
260	ORTAKÖY	3+1	90.000,00 TL

Page 1 of 1

**Figure 3.54** House to Let Advertisements Form

### 3.13.3 Villa for Sale Advertisements Form

Using this form you can print the advertisements about that villa for sale.



**ESER PROPERTY  
VILLAS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
320	ORTAKÖY	DUBLEX	92.000,00 TL
225	ERYAMAN	DUBLEX	98.000,00 TL

Page 1 of 1

**Figure 3.55** Villa for Sale Advertisements Form

3.13.4 Shop to Let Advertisements Form

Using this form you can print the advertisements about that shop to let.

Print Preview

Close

# ESER PROPERTY SHOPS TO LET

TELEPHONE : 0000 000 00 00 9999 999 99 99

SQUARE METER	DISTRICT	TYPE	PRICE
55	DEREBOYU	PASAJ	45.000,00 TL
85	YENIKENT	GALERI	45.000,00 TL

Page 1 of 1

Figure 3.56 Shop to Let Advertisements Form



### 3.13.5 Plot for Sale Advertisements Form

Using this form you can print the advertisements about that plot for sale.

**ESER PROPERTY**

**PLOTS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
320	SEFAKÖY	YOLÜSTÜ	85.000,00 TL
400	LEMAR	PARKYANI	95.000,00 TL

Page 1 of 1

**Figure 3.57** Plot for Sale Advertisements Form

### 3.13.6 House for Sale Advertisements Form

Using this form you can print the advertisements about that house for sale.

**Print Preview**

ESER PROPERTY  
HOUSES FOR SALE

TELEPHONE : 0000 000 00 00 9999 999 99 99

SQUARE METER	DISTRICT	TYPE	PRICE
225	TAŞKINKÖY	2+1	85.000,00 TL
145	HAMİTKÖY	3+1	66.000,00 TL

Page 1 of 1

**Figure 3.58** House for Sale Advertisements Form

### 3.13.7 Field for Sale Advertisements Form

Using this form you can print the advertisements about that field for sale.

The image shows a 'Print Preview' window for a form titled 'ESER PROPERTY FIELDS FOR SALE'. The form includes a telephone number and a table of property listings. The table has four columns: SQUARE METER, DISTRICT, TYPE, and PRICE. The data is as follows:

SQUARE METER	DISTRICT	TYPE	PRICE
3400	KANLIDERE	217 ELMA	88.000,00 TL
4600	TAŞLIKÖY	KUYUSUZ	75.000,00 TL
4600	TAŞLIKÖY	105 KAYISI	78.000,00 TL
2500	DEĞİRMELİK	KUYULU	75.000,00 TL

Page 1 of 1

**Figure 3.59** Field for Sale Advertisements Form



### 3.13.8 Shop for Sale Advertisements Form

Using this form you can print the advertisements about that shop for sale.

**ESER PROPERTY  
SHOPS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
55	GİRNEKAPI	ZEMİN	38.000,00 TL
75	KÜÇÜKKAYMAKLI	ZEMİN	65.000,00 TL

Page 1 of 1

**Figure 3.60** Shop for Sale Advertisements Form

3.13.9 Garden for Sale Advertisements Form

Using this form you can print the advertisements about that garden for sale.

**Print Preview**

Close

# ESER PROPERTY GARDENS FOR SALE

TELEPHONE : 0000 000 00 00 9999 999 99 99

SQUARE METER	DISTRICT	PRICE
220	GÖNYELİ	92.000,00 TL
360	KAYALI	88.000,00 TL

Page 1 of 1

Figure 3.61 Garden for Sale Advertisements Form

### 3.13.10 Building for Sale Advertisements Form

Using this form you can print the advertisements about that building for sale.

**ESER PROPERTY**

**BUILDINGS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
320	METROPOL	4+1	75.000,00 TL
185	BAHÇELİEVLER	3+1	98.765,00 TL

Page 1 of 1

**Figure 3.62** Building for Sale Advertisements Form



### 3.13.11 Farm for Sale Advertisements Form

Using this form you can print the advertisements about that farm for sale.

**ESER PROPERTY  
FARMS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

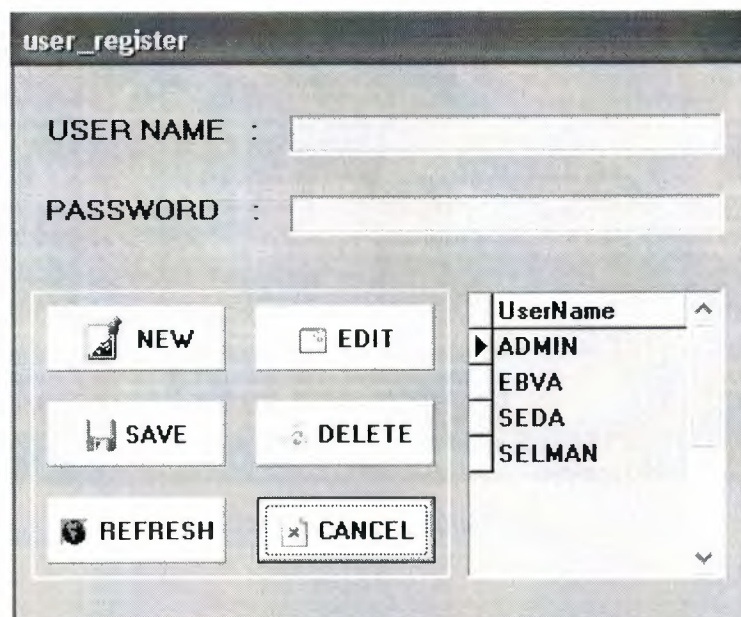
SQUARE METER	DISTRICT	TYPE	PRICE
3200	HAMITKÖY	BÜYÜKBAŞ	87.000,00 TL
4000	DEMİRHAN	TAVUK	65.000,00 TL

Page 1 of 1

**Figure 3.63** Farm for Sale Advertisements Form

### 3.14 User Register Menu

When you press the user register button you are going to open new form here who will use this program can be registered and they can use the program and same time you can exchange your password.If you press new button new admin can be added to user list.If you press edit button you can change your informations.If you press save button you can save your informations.If you press delete button you can deleted.User information from system.If you press refresh button you can clean the page.If you press cancel button you can leave this form and you can go to main menu.



The screenshot shows a window titled "user\_register". It has two text input fields: "USER NAME" and "PASSWORD". Below the inputs are six buttons arranged in a 3x2 grid: "NEW" (with a plus icon), "EDIT" (with a document icon), "SAVE" (with a floppy disk icon), "DELETE" (with a trash can icon), "REFRESH" (with a circular arrow icon), and "CANCEL" (with an 'X' icon). To the right of the buttons is a list box titled "UserName" with a scroll bar. The list contains four items: "ADMIN", "EBVA", "SED", and "SELMAN".

Figure 3.64 User Register From

### 3.15 About Menu

This form gives informations about the current program and owner of this program.

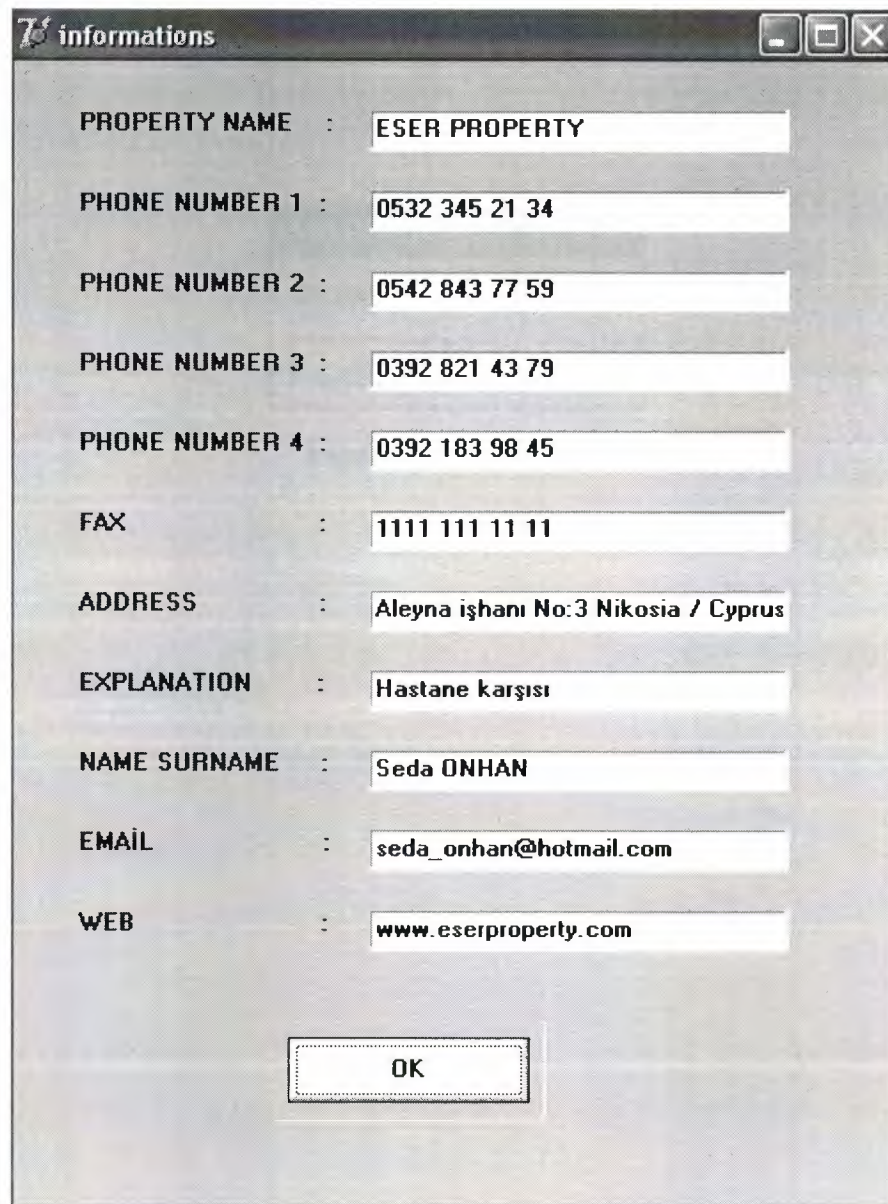


**Figure 3.65** About Form



### 3.16 Informations Menu

Using informations menu you can get all informations about the property.



The screenshot shows a window titled 'informations' with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a form with the following fields:

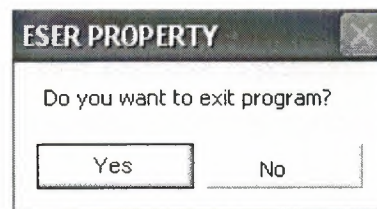
Field	Value
PROPERTY NAME	ESER PROPERTY
PHONE NUMBER 1	0532 345 21 34
PHONE NUMBER 2	0542 843 77 59
PHONE NUMBER 3	0392 821 43 79
PHONE NUMBER 4	0392 183 98 45
FAX	1111 111 11 11
ADDRESS	Aleyna işhanı No:3 Nikosia / Cyprus
EXPLANATION	Hastane karşısı
NAME SURNAME	Seda ONHAN
EMAIL	seda_onhan@hotmail.com
WEB	www.eserproperty.com

At the bottom center of the window is an 'OK' button.

Figure 3.66 Informations Form

### 3.17 Exit Menu

When you click the exit menu ( yes / no ) you can decide to continue search or exit the program.



**Figure 3.67** Exit Form

## CONCLUSION

In this Graduation Project stock program for any property using Delphi was examined.

This program can be used easily for each user that can record customer information.

The operation structures of this program could be explained briefly; as follows when user executes program, first database connection screen appears. In this screen user enters user name and password to use the program. so user must have a valid user name and password. Also user must have appropriate privileges on database; such as view, add, update, delete.

When the user name and password are entered correctly user meets the Main Menu screen. As you can see in this figure there are 15 sections; house to let, house for sale, shop to let, shop for sale, plot for sale, garden for sale, building for sale, farm for sale, villa for sale, field for sale, flier print, about, informations, user register and exit are the names of the sections.

For future implementations the current program can be developed using different program languages.



## REFERENCES

<http://www.codegear.com>

[http://www.scalabium.com/faq/dc\\_tips.htm](http://www.scalabium.com/faq/dc_tips.htm)

<http://www.nevrona.com/>

Delphi Programming Explorer, Jeff Dontemann – Jim Mischel ISBN 1-883-57725-X

Database Application Developers Book for Delphi (e Book)

Borland Delphi 6 for Windows (e Book)

Mastering Delphi 6 – Marco Cantu

## APPENDIX

### Program Codes

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, jpeg, ExtCtrls, ImgList, ComCtrls, ToolWin;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Label1: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn7: TBitBtn;
    BitBtn8: TBitBtn;
    BitBtn9: TBitBtn;
    BitBtn10: TBitBtn;
    BitBtn11: TBitBtn;
    BitBtn12: TBitBtn;
    BitBtn13: TBitBtn;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    Bevel5: TBevel;
    Bevel6: TBevel;
    Bevel7: TBevel;
    Bevel8: TBevel;
    Bevel9: TBevel;
    Bevel10: TBevel;
    Bevel11: TBevel;
    Bevel12: TBevel;
    Timer1: TTimer;
    ImageList1: TImageList;
    ImageList2: TImageList;
    Label2: TLabel;
    BitBtn14: TBitBtn;
    Bevel13: TBevel;
    Bevel14: TBevel;
    procedure BitBtn1Click(Sender: TObject);
```

```

procedure BitBtn2Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure BitBtn5Click(Sender: TObject);
procedure BitBtn6Click(Sender: TObject);
procedure BitBtn9Click(Sender: TObject);
procedure BitBtn7Click(Sender: TObject);
procedure BitBtn8Click(Sender: TObject);
procedure BitBtn10Click(Sender: TObject);
procedure BitBtn11Click(Sender: TObject);
procedure BitBtn12Click(Sender: TObject);
procedure BitBtn13Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure Label2DbClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure BitBtn14Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

  uses unit2, unit5, unit8, unit10, unit6, unit3, unit4, unit9, unit7,
    unit11, unit22, unit23, unit44, unit45, unit47;

{$R *.dfm}

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  housetolet.ShowModal;
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
  shoptolet.ShowModal;
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
begin
  plot.ShowModal;
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
begin
  building.ShowModal;
end;

```



```

end;

procedure TForm1.BitBtn5Click(Sender: TObject);
begin
villa.ShowModal;
end;

procedure TForm1.BitBtn6Click(Sender: TObject);
begin
flierprint.ShowModal;
end;

procedure TForm1.BitBtn9Click(Sender: TObject);
begin
garden.ShowModal;
end;

procedure TForm1.BitBtn7Click(Sender: TObject);
begin
houseforsale.ShowModal;
end;

procedure TForm1.BitBtn8Click(Sender: TObject);
begin
shopforsale.ShowModal;
end;

procedure TForm1.BitBtn10Click(Sender: TObject);
begin
farm.ShowModal;
end;

procedure TForm1.BitBtn11Click(Sender: TObject);
begin
field.ShowModal;
end;

procedure TForm1.BitBtn12Click(Sender: TObject);
begin
about.ShowModal;
end;

procedure TForm1.BitBtn13Click(Sender: TObject);
begin
if(Application.MessageBox('Do you want to exit program?','ESER
PROPERTY',MB_YESNO)=IDYES)then
halt;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin

```

```
form1.caption:='ESER PROPERTY      '+DateTOStr(now)+'      '+TIMETostr(now)+'      ';
end;
```

```
procedure TForm1.Label2DbClick(Sender: TObject);
begin
informations.Show;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
Form1.ClientHeight:=599;
Form1.ClientWidth:=1072;
end;
```

```
procedure TForm1.BitBtn14Click(Sender: TObject);
begin
form47.ShowModal;
end;
```

```
end.
```

```
unit Unit2;
```

```
interface
```

```
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, DBCtrls, ExtCtrls, Buttons, ImgList, ComCtrls, ToolWin,
Mask;
```

```
type
Thousetolet = class(TForm)
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
ToolBar1: TToolBar;
ToolButton1: TToolButton;
ToolButton2: TToolButton;
ToolButton3: TToolButton;
ToolButton4: TToolButton;
ToolButton5: TToolButton;
ToolButton6: TToolButton;
ToolButton7: TToolButton;
ToolButton8: TToolButton;
ToolButton9: TToolButton;
ToolButton10: TToolButton;
ToolButton11: TToolButton;
ImageList1: TImageList;
```

```

BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
Bevel1: TBevel;
Bevel2: TBevel;
DBCheckBox1: TDBCheckBox;
Bevel3: TBevel;
Bevel4: TBevel;
Bevel5: TBevel;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure ToolButton9Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure FormCreate(Sender: TObject);

```



```

private
  { Private declarations }
public
  { Public declarations }
end;

var
  housetolet: Thousetolet;

implementation

  uses unit45, unit12, unit24;

  {$R *.dfm}

  procedure Thousetolet.ToolButton1Click(Sender: TObject);
  begin
    dm.tkhouse.Insert;
  end;

  procedure Thousetolet.ToolButton3Click(Sender: TObject);
  begin
    dm.tkhouse.Prior;
  end;

  procedure Thousetolet.ToolButton5Click(Sender: TObject);
  begin
    dm.tkhouse.Next;
  end;

  procedure Thousetolet.ToolButton7Click(Sender: TObject);
  begin
    dm.tkhouse.Cancel;
  end;

  procedure Thousetolet.BitBtn1Click(Sender: TObject);
  begin
    housetoletsearch.ShowModal;
  end;

  procedure Thousetolet.BitBtn2Click(Sender: TObject);
  begin
    housetoletreport.QuickRep1.Preview;
  end;

  procedure Thousetolet.DBCheckBox1Click(Sender: TObject);
  begin
    if DBCheckBox1.Checked=true then
    begin
      GroupBox2.Visible:=true;
      DBCheckBox1.Caption:='Already rented';
    end;
  end;

```

```

end;
if DBCheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='this house give to let';
end;
end;

procedure Thousetolet.ToolButton11Click(Sender: TObject);
begin
dm.tkhouse.Edit;
dm.tkhouse.Post;
ShowMessage('Record is registered');
end;

procedure Thousetolet.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Thousetolet.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Thousetolet.ToolButton9Click(Sender: TObject);
begin
try
if(Application.MessageBox('Record will be delete are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tkhouse.Delete;
except
ShowMessage('Cant delete empty record!');
end;
end;

procedure Thousetolet.FormKeyPress(Sender: TObject; var Key: Char);
begin
If (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Thousetolet.FormCreate(Sender: TObject);
begin
housetolet.ClientHeight:=606;
housetolet.ClientWidth:=695;
end;

```

end.

unit Unit3;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, ImgList, ComCtrls, ToolWin, StdCtrls, Mask, DBCtrls, Buttons,  
ExtCtrls;

type

Tshopforsale = class(TForm)  
  ToolBar1: TToolBar;  
  ToolButton1: TToolButton;  
  ToolButton2: TToolButton;  
  ToolButton3: TToolButton;  
  ToolButton4: TToolButton;  
  ToolButton5: TToolButton;  
  ToolButton6: TToolButton;  
  ToolButton7: TToolButton;  
  ToolButton8: TToolButton;  
  ToolButton9: TToolButton;  
  ToolButton10: TToolButton;  
  ToolButton11: TToolButton;  
  ImageList1: TImageList;  
  Bevel1: TBevel;  
  Bevel2: TBevel;  
  Bevel3: TBevel;  
  BitBtn1: TBitBtn;  
  BitBtn2: TBitBtn;  
  Bevel4: TBevel;  
  Bevel5: TBevel;  
  DBCheckBox1: TDBCheckBox;  
  GroupBox1: TGroupBox;  
  GroupBox2: TGroupBox;  
  GroupBox3: TGroupBox;  
  Label1: TLabel;  
  Label2: TLabel;  
  Label3: TLabel;  
  Label4: TLabel;  
  Label5: TLabel;  
  Label6: TLabel;  
  Label7: TLabel;  
  Label8: TLabel;  
  Label9: TLabel;  
  Label10: TLabel;  
  Label11: TLabel;  
  Label12: TLabel;



```

Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
procedure DBCheckBox1Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  shopforsale: Tshopforsale;

implementation

  uses unit45, unit15, unit27;

{$R *.dfm}

procedure Tshopforsale.DBCheckBox1Click(Sender: TObject);
begin
  if DBCheckBox1.Checked=true then
  begin
    GroupBox2.Visible:=true;

```

```

DBCheckBox1.Caption:='Already sold';
end;
if DBCheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell shop';
end;
end;

procedure Tshopforsale.ToolButton11Click(Sender: TObject);
begin
dm.tsshop.Edit;
dm.tsshop.Post;
ShowMessage('Record is registered');
end;

procedure Tshopforsale.ToolButton1Click(Sender: TObject);
begin
dm.tsshop.Insert;
end;

procedure Tshopforsale.ToolButton3Click(Sender: TObject);
begin
dm.tsshop.Prior;
end;

procedure Tshopforsale.ToolButton5Click(Sender: TObject);
begin
dm.tsshop.Next;
end;

procedure Tshopforsale.ToolButton7Click(Sender: TObject);
begin
dm.tsshop.Cancel;
end;

procedure Tshopforsale.BitBtn1Click(Sender: TObject);
begin
shopforsalesearch.ShowModal;
end;

procedure Tshopforsale.BitBtn2Click(Sender: TObject);
begin
shopforsalereport.QuickRep1.Preview;
end;

procedure Tshopforsale.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

```

```

procedure Tshopforsale.DBEdit1Exit(Sender: TObject);
begin
  if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Tshopforsale.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key = #13) then
  begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
  end;

end;

procedure Tshopforsale.ToolButton9Click(Sender: TObject);
begin
  try
    if (Application.MessageBox('Record will be deleted are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tsshop.Delete;
  except
    ShowMessage('Cant delete empty record!');
  end;

end;

procedure Tshopforsale.FormCreate(Sender: TObject);
begin
shopforsale.ClientHeight:=608;
shopforsale.ClientWidth:=695;
end;

end.

unit Unit4;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,
  ToolWin;

type
  Thouseforsale = class(TForm)
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;

```



ToolButton2: TToolButton;  
 ToolButton3: TToolButton;  
 ToolButton4: TToolButton;  
 ToolButton5: TToolButton;  
 ToolButton6: TToolButton;  
 ToolButton7: TToolButton;  
 ToolButton8: TToolButton;  
 ToolButton9: TToolButton;  
 ToolButton10: TToolButton;  
 ToolButton11: TToolButton;  
 ImageList1: TImageList;  
 Bevel1: TBevel;  
 Bevel2: TBevel;  
 BitBtn1: TBitBtn;  
 BitBtn2: TBitBtn;  
 DBCheckBox1: TDBCheckBox;  
 Bevel3: TBevel;  
 Bevel4: TBevel;  
 Bevel5: TBevel;  
 GroupBox1: TGroupBox;  
 GroupBox2: TGroupBox;  
 GroupBox3: TGroupBox;  
 Label1: TLabel;  
 Label2: TLabel;  
 Label3: TLabel;  
 Label4: TLabel;  
 Label5: TLabel;  
 Label6: TLabel;  
 Label7: TLabel;  
 Label8: TLabel;  
 Label9: TLabel;  
 Label10: TLabel;  
 Label11: TLabel;  
 Label12: TLabel;  
 Label13: TLabel;  
 Label14: TLabel;  
 Label15: TLabel;  
 DBEdit1: TDBEdit;  
 DBEdit2: TDBEdit;  
 DBEdit3: TDBEdit;  
 DBEdit4: TDBEdit;  
 DBEdit5: TDBEdit;  
 DBEdit6: TDBEdit;  
 DBEdit7: TDBEdit;  
 DBEdit8: TDBEdit;  
 DBEdit9: TDBEdit;  
 DBEdit10: TDBEdit;  
 DBEdit11: TDBEdit;  
 DBEdit12: TDBEdit;  
 DBEdit13: TDBEdit;  
 DBEdit14: TDBEdit;

```

DBEdit15: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  houseforsale: Thouseforsale;

implementation

  uses unit45, unit14, unit26;

{$R *.dfm}

procedure Thouseforsale.ToolButton1Click(Sender: TObject);
begin
  dm.tshouse.Insert;
end;

procedure Thouseforsale.ToolButton3Click(Sender: TObject);
begin
  dm.tshouse.Prior;
end;

procedure Thouseforsale.ToolButton5Click(Sender: TObject);
begin
  dm.tshouse.Next;
end;

procedure Thouseforsale.ToolButton7Click(Sender: TObject);
begin
  dm.tshouse.Cancel;
end;

procedure Thouseforsale.ToolButton11Click(Sender: TObject);
begin

```

```

dm.tshouse.Edit;
dm.tshouse.Post;
ShowMessage('Record is registered');
end;

procedure Thouseforsale.BitBtn1Click(Sender: TObject);
begin
houseforsalesearch.ShowModal;
end;

procedure Thouseforsale.BitBtn2Click(Sender: TObject);
begin
houseforsalereport.QuickRep1.Preview;
end;

procedure Thouseforsale.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already sold';
end;
if DBcheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell house';
end;
end;

procedure Thouseforsale.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Thouseforsale.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Thouseforsale.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

procedure Thouseforsale.ToolButton9Click(Sender: TObject);
begin

```



```

try
if (Application.MessageBox('Record wii be deleted are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tshouse.Delete;
except
  ShowMessage('Cant delete empty record!');
end;

```

```

end;

```

```

procedure Thouseforsale.FormCreate(Sender: TObject);
begin
houseforsale.ClientHeight:=609;
houseforsale.ClientWidth:=695;
end;

```

```

end.

```

```

unit Unit5;

```

```

interface

```

```

uses

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,
  ToolWin;

```

```

type

```

```

Tshoptolet = class(TForm)
  ToolBar1: TToolBar;
  ToolButton1: TToolButton;
  ToolButton2: TToolButton;
  ToolButton3: TToolButton;
  ToolButton4: TToolButton;
  ToolButton5: TToolButton;
  ToolButton6: TToolButton;
  ToolButton7: TToolButton;
  ToolButton8: TToolButton;
  ToolButton9: TToolButton;
  ToolButton10: TToolButton;
  ToolButton11: TToolButton;
  ImageList1: TImageList;
  Bevel1: TBevel;
  Bevel2: TBevel;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  DBCheckBox1: TDBCheckBox;
  Bevel3: TBevel;

```

```

Bevel4: TBevel;
Bevel5: TBevel;
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton9Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public

```

```

    { Public declarations }
end;

var
    shoptolet: Tshoptolet;

implementation

    uses unit45, unit13, unit25;

    {$R *.dfm}

    procedure Tshoptolet.ToolButton1Click(Sender: TObject);
    begin
        dm.tkshop.Insert;
    end;

    procedure Tshoptolet.ToolButton3Click(Sender: TObject);
    begin
        dm.tkshop.Prior;
    end;

    procedure Tshoptolet.ToolButton5Click(Sender: TObject);
    begin
        dm.tkshop.Next;
    end;

    procedure Tshoptolet.ToolButton7Click(Sender: TObject);
    begin
        dm.tkshop.Cancel;
    end;

    procedure Tshoptolet.ToolButton9Click(Sender: TObject);
    begin
        try
            if(Application.MessageBox('Record will be deleted are you
            sure?', 'Confirmation', MB_YESNO)=IDYES) then
                dm.tkshop.Delete;
            except

                ShowMessage('Cant delete empty record!');
            end;
        end;

    procedure Tshoptolet.BitBtn1Click(Sender: TObject);
    begin
        shoptoletsearch.ShowModal;
    end;

    procedure Tshoptolet.BitBtn2Click(Sender: TObject);

```



```

begin
shoptoletreport.QuickRep1.Preview;
end;

procedure Tshoptolet.ToolButton11Click(Sender: TObject);
begin
dm.tkshop.Edit;
dm.tkshop.Post;
ShowMessage('Record is registered');
end;

procedure Tshoptolet.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tshoptolet.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Tshoptolet.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Tshoptolet.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already rented';
end;
if DBCheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='This shop give to let';
end;
end;

procedure Tshoptolet.FormCreate(Sender: TObject);
begin
shoptolet.ClientHeight:=614;
shoptolet.ClientWidth:=695;
end;

end.

```

unit Unit6;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,  
ToolWin;

type

Tplot = class(TForm)  
  ToolBar1: TToolBar;  
  ToolButton1: TToolButton;  
  ToolButton2: TToolButton;  
  ToolButton3: TToolButton;  
  ToolButton4: TToolButton;  
  ToolButton5: TToolButton;  
  ToolButton6: TToolButton;  
  ToolButton7: TToolButton;  
  ToolButton8: TToolButton;  
  ToolButton9: TToolButton;  
  ToolButton10: TToolButton;  
  ToolButton11: TToolButton;  
  ImageList1: TImageList;  
  Bevel1: TBevel;  
  Bevel2: TBevel;  
  BitBtn1: TBitBtn;  
  BitBtn2: TBitBtn;  
  DBCheckBox1: TDBCheckBox;  
  Bevel3: TBevel;  
  Bevel4: TBevel;  
  Bevel5: TBevel;  
  GroupBox1: TGroupBox;  
  GroupBox2: TGroupBox;  
  Label1: TLabel;  
  Label2: TLabel;  
  Label3: TLabel;  
  Label4: TLabel;  
  Label5: TLabel;  
  Label6: TLabel;  
  DBEdit1: TDBEdit;  
  DBEdit2: TDBEdit;  
  DBEdit3: TDBEdit;  
  DBEdit4: TDBEdit;  
  DBEdit5: TDBEdit;  
  DBEdit6: TDBEdit;  
  GroupBox3: TGroupBox;  
  Label7: TLabel;

```

Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  plot: Tplot;

implementation

  uses unit45, unit16, unit28;

{$R *.dfm}

procedure Tplot.ToolButton1Click(Sender: TObject);
begin
  dm.tsplot.Insert;
end;

procedure Tplot.ToolButton3Click(Sender: TObject);
begin
  dm.tsplot.Prior;
end;

procedure Tplot.ToolButton5Click(Sender: TObject);
begin

```





**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**Stock Property by Using Delphi**

**Graduation Project**

**COM 400**

**Student: Seda ONHAN (20032905)**

**Supervisor: Assist. Prof. Dr. Imanov ELBRUS**

**Nicosia – 2008**





## ACKNOWLEDGEMENT

*First of all, I would like give my special thanks to my supervisor Assist. Prof. Dr. Imanov ELBRUS. He helped and supported me to complete my project by any means of necessary. In addition to this he never doubted about me, he always believed in me that I will fulfill and succeed on my project. I am glad to that I did not disappoint him.*

*Furthermore, I want to give my special thanks and best regards to my parents. They were always kind and patient to me. I wouldn't be here without their endless support.*

*Finally, I want to give my special thanks to my friends whose are Cemal Kavalcioğlu, Selman Oğuzhan ESER. They are supported and helped me to complete my project. I am very happy to have such friends.*

## **ABSTRACT**

The aim of this Project is to record the stock device for any Properties Company. The program was prepared by using Delphi 7 programming and using Paradox7. Delphi is a programming language that can be used with Paradox7.

This project consists of many different pages but most of them depended each other Initially, SIGN IN form comes to screen. Afterwards the Main menu of Properties Company comes to screen. After Main Menu you are going to see the main form that contains 15 main menus.



## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENT .....</b>	<b>I</b>
<b>ABSTRACT.....</b>	<b>II</b>
<b>TABLE OF CONTENTS .....</b>	<b>III</b>
<b>INTRODUCTION .....</b>	<b>1</b>

### CHAPTER ONE : BASIC CONCEPT OF DELPHI

1.1 Introduction to Delphi.....	2
1.2 What is Delphi? .....	2
1.2.1 Delphi Compilers .....	2
1.2.2 What kind of programming can you do with Delphi? .....	3
1.2.3 History of Delphi .....	4
1.2.4 Advantages & Disadvantages Delphi .....	6
1.3 Delphi 6 Editions .....	7
1.3.1 Delphi 6 Architect.....	7
1.3.2 Installation Delphi 6.....	8
1.4 A Tour of the Environment.....	10
1.4.1 Running Delphi for the First Time .....	10
1.4.2 The Delphi IDE.....	11
1.4.3 The Menus & Toolbar.....	12
1.4.4 The Component Palette.....	12
1.4.5 The Code Editor.....	13
1.4.6 The Object Inspector.....	14
1.4.7 The Object TreeView.....	15
1.4.8 Class Completion.....	16
1.4.9 Debugging applications .....	17
1.4.10 Exploring Databases .....	18
1.4.11 Templates and the Object Repository .....	19
1.5 Programming with Delphi .....	20
1.5.1 Starting a New Application.....	20
1.5.2 Setting Property Values .....	21
1.5.3 Adding objects to the form .....	22
1.5.4 Add a Table and a StatusBar to the Form.....	22
1.5.5 Connecting to a Database .....	24

### CHAPTER TWO : THE RAVE REPORTING

2.1 Project Tree.....	28
2.2 Design Tools .....	29
2.3 Reuse and Maintenance Tools .....	32
2.4 Standard Components .....	34
2.5 Drawing Components .....	35
2.6 Reporting Components .....	35
2.7 Barcode Components .....	39
2.8 Anchors .....	39

2.9	Code Based Reports .....	40
2.9.1	Simple Code Base Report .....	40
2.9.2	Tabular Code Based Report .....	41
2.9.3	Graphical Code Based Report.....	43
2.10	Visually Designed Reports .....	45
2.10.1	The Visual Designer .....	45
2.10.2	Interacting with the Project.....	48
2.11	Data Aware Reports.....	55
2.11.1	The Database Connection .....	55
2.11.2	The Driver Data View.....	55
2.11.3	Regions and Bands.....	58
2.11.4	Adding Fields.....	60
2.11.5	Adding the Report to Your Project .....	60

## CHAPTER THREE : STOCK PROPERTY BY USING DELPHI

3.1	Database Connection Screen .....	61
3.2	Main Menu.....	63
3.3	House to Let Menu .....	64
3.3.1	House to Let Organize Form .....	64
3.3.2	House to Let Search Form .....	65
3.3.3	House to Let Report Form .....	68
3.4	House for Sale Menu .....	69
3.4.1	House for Sale Organize Form .....	69
3.4.2	House for Sale Search Form .....	70
3.4.3	Hose for Sale Report Form .....	73
3.5	Shop to Let Menu .....	74
3.5.1	Shop to Let Organize Form .....	74
3.5.2	Shop to Let Search Form .....	75
3.5.3	Shop to Let Report Form .....	78
3.6	Shop for Sale Menu .....	79
3.6.1	Shop for Sale Organize Form .....	79
3.6.2	Shop for Sale Search Form .....	80
3.6.3	Shop for Sale Report Form .....	83
3.7	Plot to Let Menu .....	84
3.7.1	Plot to Let Organize Form .....	84
3.7.2	Plot to Let Search Form .....	85
3.7.3	Plot to Let Report Form .....	88
3.8	Garden for Sale Menu.....	89
3.8.1	Garden for Sale Organize Form.....	89
3.8.2	Garden for Sale Search Form.....	90
3.8.3	Garden for Sale Report Form.....	93
3.9	Building for Sale Menu.....	94
3.9.1	Building for Sale Organize Form.....	94
3.9.2	Buildig for Sale Search Form .....	95
3.9.3	Building for Sale Report Form .....	98
3.10	Farm for Sale Menu .....	99
3.10.1	Farm for Sale Organize Form .....	99

3.10.2	Farm for Sale Search Form .....	100
3.10.3	Farm for Sale Report Form .....	103
3.11	Villa for Sale Menu .....	104
3.11.1	Villa for Sale Organize Form .....	104
3.11.2	Villa for Sale Search Form .....	105
3.11.3	Villa for Sale Report Form .....	108
3.12	Field for Sale Menu .....	109
3.12.1	Field for Sale Organize Form .....	109
3.12.2	Filed for Sale Search Form .....	110
3.12.3	Field for Sale Report Form .....	113
3.13	Flier Print Menu .....	114
3.13.1	Flier Print Organize Form .....	114
3.13.2	House to Let Advertisements Form .....	115
3.13.3	Villa for Sale Advertisements Form .....	116
3.13.4	Shop to Let Advertisements Form .....	117
3.13.5	Plot for Sale Advertisements Form .....	118
3.13.6	House for Sale Advertisements Form .....	119
3.13.7	Field for Sale Advertisements Form .....	120
3.13.8	Shop for Sale Advertisements Form .....	121
3.13.9	Garden for Sale Advertisements Form .....	122
3.13.10	Building for Sale Advertisements Form .....	123
3.13.11	Farm for Sale Advertisements Form .....	124
3.14	User Register Menu .....	125
3.15	About Menu .....	126
3.16	Informations Menu .....	127
3.17	Exit Menu .....	128
<b>CONCLUSION</b> .....		129
<b>REFERENCES</b> .....		130
<b>APPENDIX</b> .....		140

## **INTRODUCTION**

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

The project consists of the introduction, three chapters, and conclusion.

- Chapter one describes Basic Concept of Delphi.
- Chapter two describes the database that uses Delphi programming language.
- Chapter three explains Stock Property by Using Delphi.



## **CHAPTER ONE**

### **1 BASIC CONCEPT OF DELPHI**

#### **1.1 Introduction to Delphi**

Although I am not the most experienced or knowledgeable person on the forums I thought it was time to write a good introductory article for Delphi

#### **1.2 What is Delphi?**

Delphi is a Rapid Application Development (RAD) environment. It allows you to drag and drop components on to a blank canvas to create a program. Delphi will also allow you to use write console based DOS like programs.

Delphi is based around the Pascal language but is more developed object orientated derivative. Unlike Visual Basic, Delphi uses punctuation in its basic syntax to make the program easily readable and to help the compiler sort the code. Although Delphi code is not case sensitive there is a generally accepted way of writing Delphi code. The main reason for this is so that any programmer can read your code and easily understand what you are doing, because they write their code like you write yours.

For the purposes of this series I will be using Delphi 6. Delphi 6 provides all the tools you need to develop test and deploy Windows applications, including a large number of so-called reusable components.

Borland Delphi provides a cross platform solution when used with Borland Kylix – Borland's RAD tool for the Linux platform.

##### **1.2.1 Delphi Compilers**

There are two types compiler for Delphi

- Turbo Delphi: Free industrial strength Delphi RAD (Rapid Application Development) environment and compiler for Windows. It comes with 200+ components and its own Visual Component Framework.

- Turbo Delphi for .NET: Free industrial strength Delphi application development environment and compiler for the Microsoft .NET platform.

### **1.2.2 What kind of programming can you do with Delphi?**

The simple answer is “more or less anything”. Because the code is compiled, it runs quickly, and is therefore suitable for writing more or less any program that you would consider a candidate for the Windows operating system.

You probably won't be using it to write embedded systems for washing machines, toasters or fuel injection systems, but for more or less anything else, it can be used (and the chances are that probably someone somewhere has!)

Some projects to which Delphi is suited:

- Simple, single user database applications
- Intermediate multi-user database applications
- Large scale multi-tier, multi-user database applications
- Internet applications
- Graphics Applications
- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools
- Communications tools using the Internet, Telephone or LAN
- Web based applications

This is not intended to be an exhaustive list, more an indication of the depth and breadth of Delphi's applicability. Because it is possible to access any and all of the Windows API, and because if all else fails, Delphi will allow you to drop a few lines of assembler code directly into your ordinary Pascal instructions, it is possible to do more or less anything. Delphi can also be used to write Dynamically Linked Libraries (DLLs) and can call out to DLLs written in other programming languages without difficulty.

Because Delphi is based on the concept of self contained Components (elements of code that can be dropped directly on to a form in your application, and exist in object form, performing their function until they are no longer required), it is possible to build applications very rapidly. Because Delphi has been available for quite some time, the number of pre-written components has been increasing to the point that now there is a component to do more or less anything you can imagine. The job of the programmer has become one of gluing together appropriate components with code that operates them as required.

### **1.2.3 History of Delphi**

Delphi was one of the first of what came to be known as "RAD" tools, for Rapid Application Development, when released in 1995 for the 16-bit Windows 3.1. Delphi 2, released a year later, supported 32-bit Windows environments, and a C++ variant, C++ Builder, followed a few years after.

The chief architect behind Delphi, and its predecessor Turbo Pascal, was Anders Hejlsberg until he was headhunted in 1996 by Microsoft, where he worked on Visual J++ and subsequently became the chief designer of C Sharp programming language C# and a key participant in the creation of the Microsoft .NET Framework.

In 2001 a Linux version known as Kylix programming tool Kylix became available. However, due to low quality and subsequent lack of interest, Kylix was abandoned after version 3.

Support for Linux and Windows cross platform development (through Kylix and the CLX component library) was added in 2002 with the release of Delphi 6.

Delphi 8, released December 2003, was a .NET -only release that allowed developers to compile Delphi Object Pascal code into .NET Microsoft Intermediate Language MSIL. It was also significant in that it changed its IDE for the first time, from the multiple-floating-window-on-desktop style IDE to a look and feel similar to Microsoft's Visual Studio.NET.

Although Borland fulfilled one of the biggest requests from developers (.NET support), it was criticized both for making it available too late, when a lot of former Delphi developers had already moved to C#, and for focusing so much on backward compatibility that it was not very easy to write new code in Delphi. Delphi 8 also lacked significant high-level features of the c sharp, C# language, as well as many of the more appealing features of Microsoft's Visual Studio IDE. (There were also concerns about the future of Delphi Win32 development. Because Delphi 8 did not support Win32, Delphi 7.1 was included in the Delphi 8 package.)

The next version, Delphi 2005 (Delphi 9), included the Win32 and .NET development in a single IDE, reiterating Borland's commitment to Win32 developers. Delphi 2005 includes design-time manipulation of live data from a database. It also includes an improved IDE and added a "for ... in" statement (like C#'s for each) to the language. However, it was criticized by some for its bugs; both Delphi 8 and Delphi 2005 had stability problems when shipped, which were only partially resolved in service packs.

In late 2005, Delphi 2006 was released and federated development of C# and Delphi.NET, Delphi Win32 and C++ into a single IDE. It was much more stable than Delphi 8 or Delphi 2005 when shipped, and improved even more after the service packs and several hot fixes.

On February 8, 2006, Borland announced that it was looking for a buyer for its IDE and database line of products, which include Delphi, to concentrate on its Application Lifecycle Management ALM line. The news met with voluble optimism from the remaining Delphi users.

On September 6, 2006, The Developer Tools Group (the working name of the not yet spun off company) of Borland Software Corporation released single language versions of Borland Developer Studio, bringing back the popular "Turbo" moniker. The Turbo product set includes Turbo Delphi for Win32, Turbo Delphi for .NET, Turbo C++, and Turbo C#. Each version is available in two editions: "Explorer" a free downloadable version and "Professional" a relatively cheap (US\$399) version which



opens access to thousands of third-party components. Unlike earlier “Personal” editions of Delphi, new “Explorer” editions can be used for commercial development.

On November 14, 2006, Borland announced the cancellation of the sale of its Development tools; instead of that it would spin them off into an independent company named “CodeGear”

#### **1.2.4 Advantages & Disadvantages Delphi**

Delphi exhibits the following advantages:

- Rapid Application Development (RAD)
- Based on a well-designed language – high-level and strongly typed, with low-level escapes for experts
- A large community on Usenet and the World Wide Web (e.g. [news://newsgroups.borland.com](mailto:news://newsgroups.borland.com) and Borland’s web access to Delphi)
- Can compile to a single executable, simplifying distribution and reducing DLL versioning issues
- Many VCL and third-party components (usually available with full source code) and tools (documentation, debug tools, etc.)
- Quick optimizing compiler and ability to use assembler code
- Multiple platform native code from the same source code
- High level of source compatibility between versions
- Cross Kylix – a third-party toolkit which allows you to compile native Kylix/Linux applications from inside the Windows Delphi IDE, hence easily enabling dual-platform development and deployment
- Cross FBC – a sister project to Cross Kylix, which enables you to cross-compile your Windows Delphi applications to multi-platform targets – supported by the Free Pascal compiler – without ever leaving the Delphi IDE
- Class helpers to bridge functionality available natively in the Delphi RTL, but not available in a new platform supported by Delphi
- The language’s object orientation features only class- and interface-based Polymorphism in object-oriented programming polymorphism

Disadvantages:

- Limited cross-platform capability for Delphi itself. Compatibles provide more architecture/OS combinations
- Access to platform and third party libraries require header files to be translated to Pascal. This creates delays and introduces the possibilities of errors in translation.
- There are fewer published books on Delphi than on other popular programming languages such as C++ and C#
- A reluctance to break any code has lead to some convoluted language design choices, and orthogonally and predictability have suffered

### **1.3 Delphi 6 Editions**

There are 3 editions in Delphi 6:

- Delphi Personal – makes learning to develop non-commercial Windows applications fast and fun. Delphi 6 Personal makes learning Windows development easy with drag-and-drop visual programming.
- Delphi Professional – adds the tools necessary to create applications with the latest Windows® ME/2000 look-and-feel. Dramatically enhance functionality with minimal code using the power and flexibility of SOAP and XML to easily integrate Web Services into client-side applications.
- Delphi Enterprise – includes additional tools, extensive options for Internet. Delphi 6 makes next-generation e-business development with Web Services a snap.

This Program will concentrate on the Enterprise edition.

#### **1.3.1 Delphi 6 Architect**

Delphi 6 Architect is designed for professional enterprise developers who need to adapt quickly to changing business rules and manage sophisticated applications that synchronize with multiple database schemas. Delphi 2006 Architect includes an advanced ECO III framework that allows developers to rapidly deploy scalable external facing Web applications with executable state diagrams, object-relational mapping, and transparent persistence.

Delphi 6 Architect includes all of the capabilities of the Enterprise edition, and includes the complete ECO III framework, including new support for ECO State Machines powered by State Chart visual diagrams, and simultaneous persistence to multiple and mixed database servers.

- State Chart Diagrams
- Executable ECO State Machines
- Multi- and Mixed- ECO database support

### 1.3.2 Installation Delphi 6

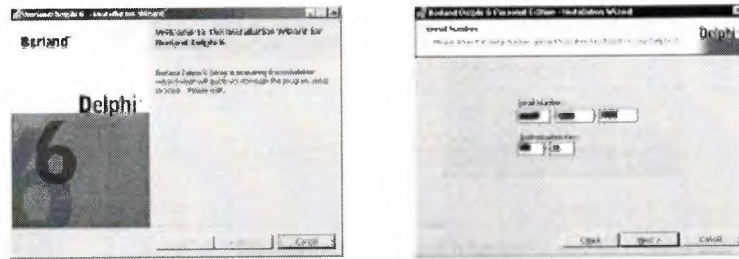
To install Delphi 6 Enterprise, run INSTALL.EXE (default location C:\Program Files\Borland Delphi) and follow the installation instructions.

We are prompted to select a product to install; you only have one choice “Delphi 6”:



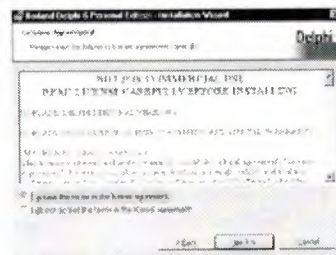
**Figure 1.1** The Select Page For Start Installation

While the setup runs, you'll need to enter your serial number and the authorization key (the two you got from inside a CdRom driver).



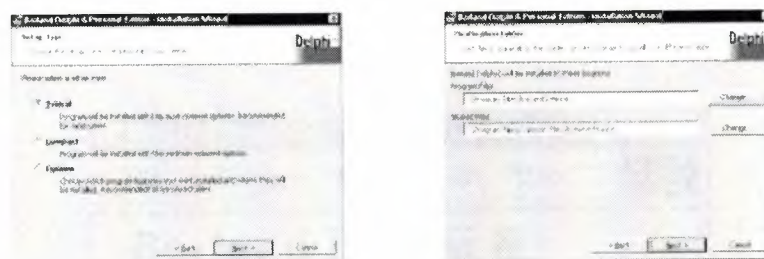
**Figure 1.2** Serial Number And Authorization Screen

Later, the License Agreement screen wills popup:



**Figure 1.3** License Agreement Screen

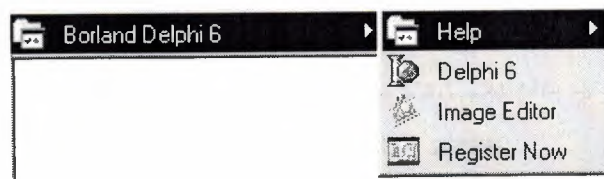
After that, you have to pick the Setup Type, choose Typical. This way Delphi 6 Enterprise will be installed with the most common options. The next screen prompts you to choose the Destination folder.



**Figure 1.4** SetUp Type and Destination Folder Screen

At the end of the installation process, the set-up program will create a sub menu in the Programs section of the Start menu, leading to the main Delphi 6 Enterprise program plus some additional tools.





**Figure 1.5** Start Menu

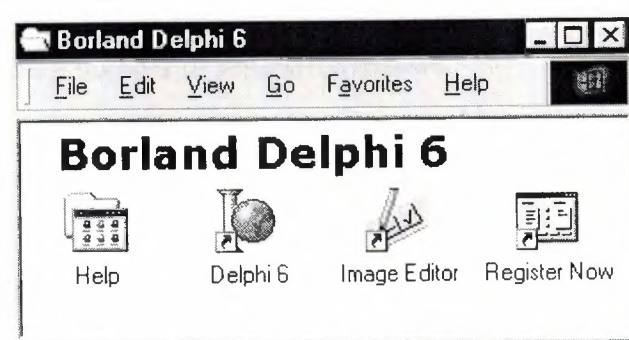
## 1.4 A Tour of the Environment

This chapter explains how to start Delphi and gives you a quick tour of the main parts and tools of the Integrated Development Environment (IDE)

### 1.4.1 Running Delphi for the First Time

You can start Delphi in a similar way to most other Windows applications:

- Choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu
- Choose Run from the Windows Start menu and type Delphi32
- Double-click Delphi32.exe in the \$(DELPHI)\Bin folder. Where \$(DELPHI) is a folder where Delphi was installed. The default is C:\Program Files\Borland\Delphi6.
- Double-click the Delphi icon on the Desktop (if you've created a shortcut)



**Figure 1.6** Borland Delphi 6 Folder

### 1.4.2 The Delphi IDE

As explained before, one of the ways to start Delphi is to choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu.

When Delphi starts (it could even take one full minute to start – depending on your hardware performance) you are presented with the IDE: the user interface where you can design, compile and debug your Delphi projects.

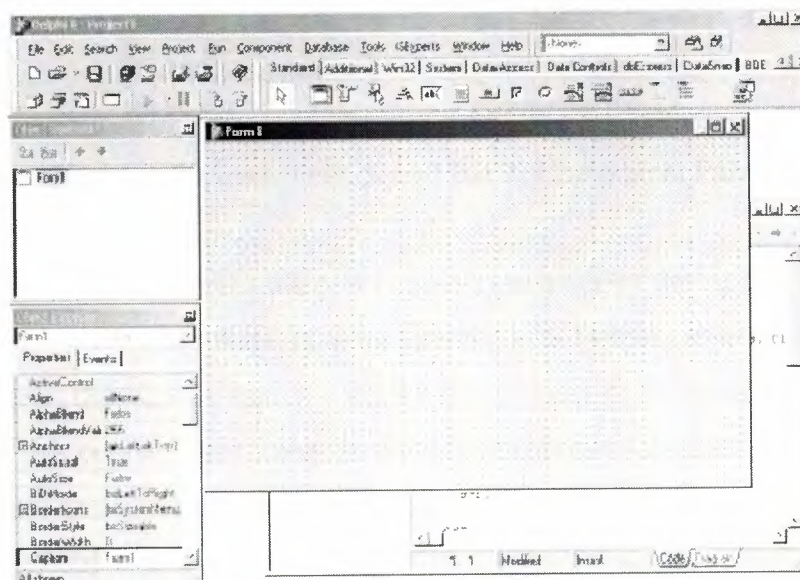


Figure 1.7 IDE

Like most other development tools (and unlike other Windows applications), Delphi IDE comprises a number of separate windows.

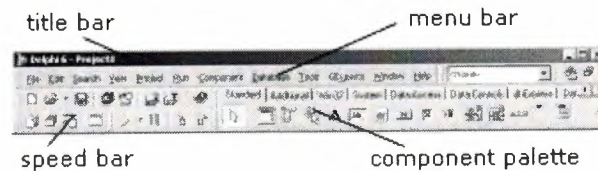
Some of the facilities that are included in the “Integrated Development Environment” (IDE) are listed below:

- A syntax sensitive program file editor
- A rapid optimizing compiler
- Built in debugging /tracing facilities
- A visual interface developer
- Syntax sensitive help files
- Database creation and editing tools

- Image/Icon/Cursor creation / editing tools
- Version Control CASE tools

### 1.4.3 The Menus & Toolbar

The main window, positioned on the top of the screen, contains the main menu, toolbar and Component palette.



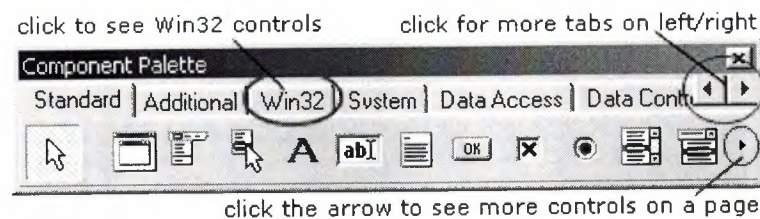
**Figure 1.8** Menu, Title, Speed Bar & Component Palette

The title bar of the main window contains the name of the current project (you'll see in some of the future chapters what exactly is a Delphi project). The menu bar includes a dozen drop-down menus – we'll explain many of the options in these menus later through this course. The toolbar provides a number of shortcuts to most frequently used operations and commands – such as running a project, or adding a new form to a project. To find out what particular button does, point your mouse “over” the button and wait for the tool tip. As you can see from the tool tip (for example, point to [Toggle Form/Unit]), many tool buttons have keyboard shortcuts ([F12]).

The menus and toolbars are freely customizable. I suggest you to leave the default arrangement while working through the chapters of this course.

### 1.4.4 The Component Palette

You are probably familiar with the fact that any window in a standard Windows application contains a number of different (visible or not to the end user) objects, like: buttons, text boxes, radio buttons, check boxes etc. In Delphi programming terminology such objects are called controls (or components). Components are the building blocks of every Delphi application. To place a component on a window you drag it from the component palette. Each component has specific attributes that enable you to control your application at design and run time.



**Figure 1.9** Component Palates

Depending on the version of Delphi (assumed Delphi 6 Personal through this course), you start with more than 85 components at your disposal – you can even add more components later (those that you create or from a third party component vendor).

The components on the Component Palette are grouped according to the function they perform. Each page tab in the Component palette displays a group of icons representing the components you can use to design your application interface. For example, the Standard and Additional pages include controls such as an edit box, a button or a scroll box.

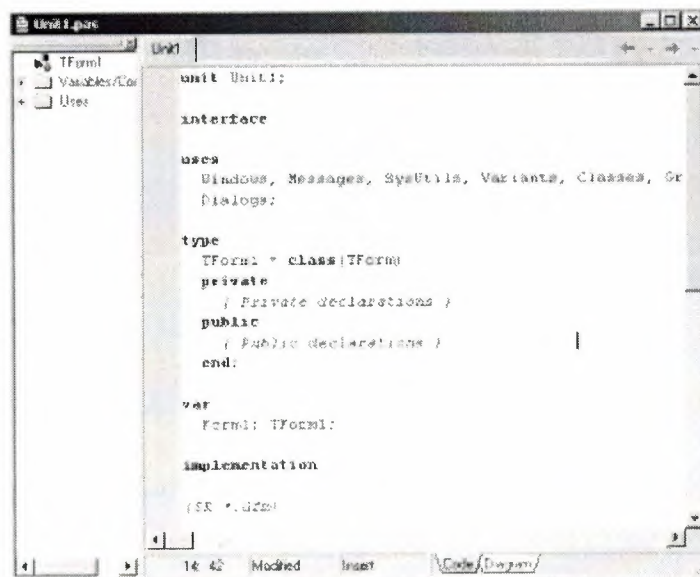
To see all components on a particular page (for example on the Win32 page) you simply click the tab name on the top of the palette. If a component palette lists more components that can be displayed on a page an arrow will appear on a far right side of the page allowing you to click it to scroll right. If a component palette has more tabs (pages) that can be displayed, more tabs can be displayed by clicking on the arrow buttons on the right-hand side.

#### **1.4.5 The Code Editor**

Each time you start Delphi, a new project is created that consists of one \*empty\* window. A typical Delphi application, in most cases, will contain more than one window – those windows are referred to as forms.

In our case this form has a name, it is called Form1. This form can be renamed, resized and moved, it has a caption and the three standard buttons which are minimize, maximize and close. As you can see a Delphi form is a regular Windows window



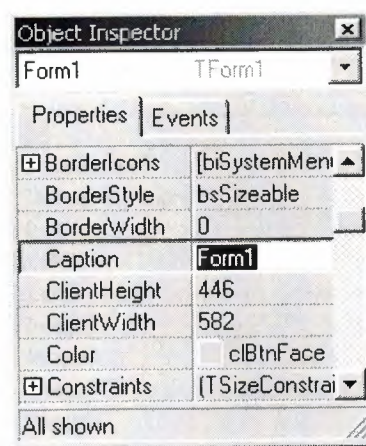


**Figure 1.10** Code Editor Window

If the Form1 is the active window and you press [F12], the Code Editor window will be placed on top. As you design user interface of your application, Delphi automatically generates the underlying Object Pascal code. More lines will be added to this window as you add your own code that drives your application. This window displays code for the current form (Form1); the text is stored in a (so-called) unit – Unit1. You can open multiple files in the Code Editor. Each file opens on a new page of the Code editor, and each page is represented by a tab at the top of the window.

#### **1.4.6 The Object Inspector**

Each component and each form has a set of properties – such as color, size, position, caption – that can be modified in the Delphi IDE or in your code, and a collection of events – such as a mouse click, keypress, or component activation – for which you can specify some additional behavior. The Object Inspector displays the properties and events (note the two tabs) for the selected component and allows you to change the property value or select the response to some event.



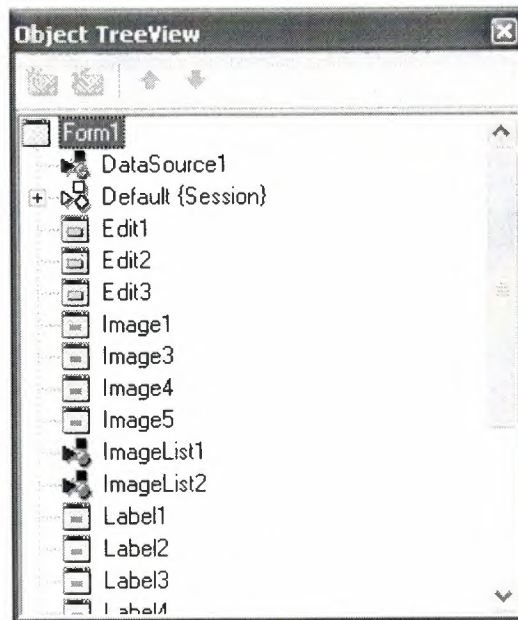
**Figure 1.11** Object Inspector

For example, each form has a Caption (the text that appears on it's title bar). To change the captions of Form1 first activate the form by clicking on it. In the Object Inspector find the property Caption (in the left column), note that it has the 'Form1' value (in the right column). To change the captions of the form simply type the new text value, like 'My Form' (without the single quotes). When you press [Enter] the caption of the form will change to My Form.

Note that some properties can be changed more simply, the position of the form on the screen can be set by entering the value for the Left and Top properties – or the form can be simply dragged to the desired location.

#### **1.4.7 The Object TreeView**

Above the Object Inspector you should see the Object TreeView window. For the moment its display is pretty simple. As you add components to the form, you'll see that it displays a component's parent-child relationships in a tree diagram. One of the great features of the Object TreeView is the ability to drag and drop components in order to change a component container without losing connections with other components.



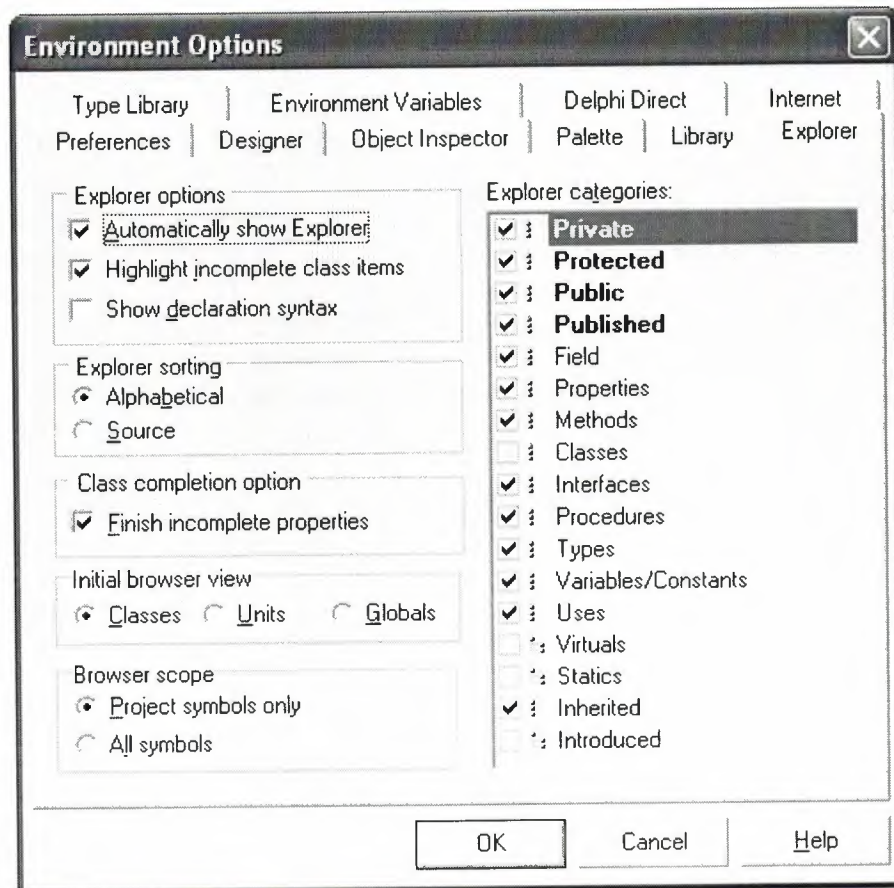
**Figure 1.12** Object Tree View

The Object TreeView, Object Inspector and the Form Designer (the Form1 window) work cooperatively. If you have an object on a form (we have not placed any yet) and click it, its properties and events are displayed in the Object Inspector and the component becomes focused in the Object TreeView.

#### **1.4.8 Class Completion**

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press Ctrl+Shift+C, or right-click and select Complete Class at Cursor. Delphi automatically adds private read and write specifies to the declarations for any properties that require them, and then creates skeleton code for all the class's methods. You can also use Class Completion to fill in class declarations for methods you've already implemented.

To configure Class Completion, choose Tools | Environment Options and click the Explorer tab.

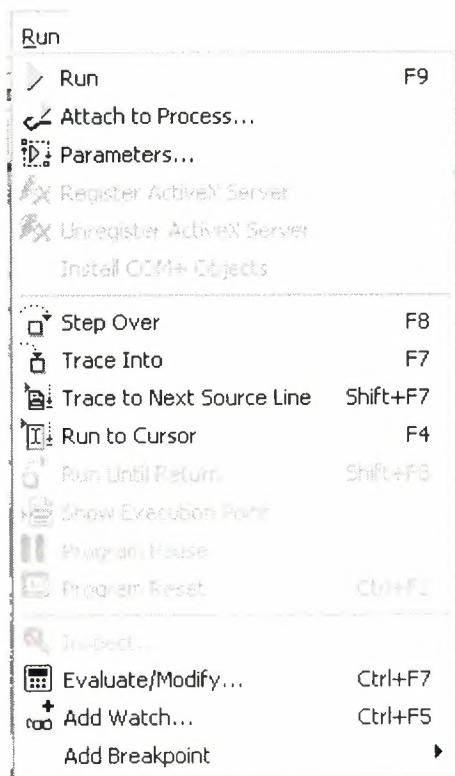


**Fig.1.13** Class Completion

#### 1.4.9 Debugging applications

The IDE includes an integrated debugger that helps you locate and fix errors in your code. The debugger lets you control program execution, watch variables, and modify data values while your application is running. You can step through your code line by line, examining the state of the program at each breakpoint.





**Figure1.14 Run**

To use the debugger, you must compile your program with debug information. Choose Project | Options, select the Compiler page, and check Debug Information. Then you can begin a debugging session by running the program from the IDE. To set debugger options, choose Tools | Debugger Options.

Many debugging windows are available, including Breakpoints, Call Stack, Watches, Local Variables, Threads, Modules, CPU, and Event Log. Display them by choosing View | Debug Windows. To learn how to combine debugging windows for more convenient use, see “Docking tool windows”.

#### **1.4.10 Exploring Databases**

The SQL Explorer (or Database Explorer in some editions of Delphi) lets you work directly with a remote database server during application development. For example, you can create, delete, or restructure tables, and you can import constraints while you are developing a database application.

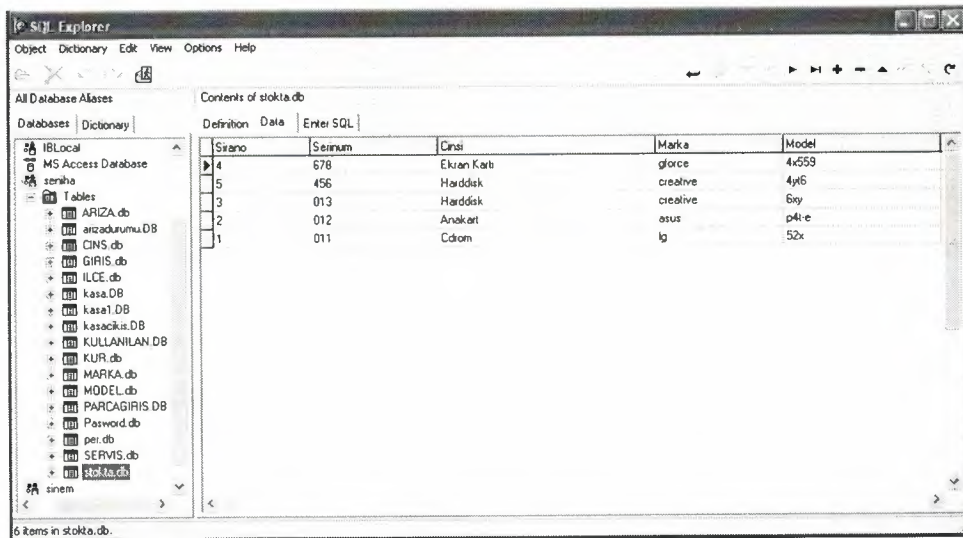


Figure 1.15 SQL Explorer

#### 1.4.11 Templates and the Object Repository

The Object Repository contains forms, dialog boxes, data modules, wizards, DLLs, sample applications, and other items that can simplify development. Choose File | New to display the New Items dialog when you begin a project. Check the Repository to see if it contains an object that resembles one you want to create.

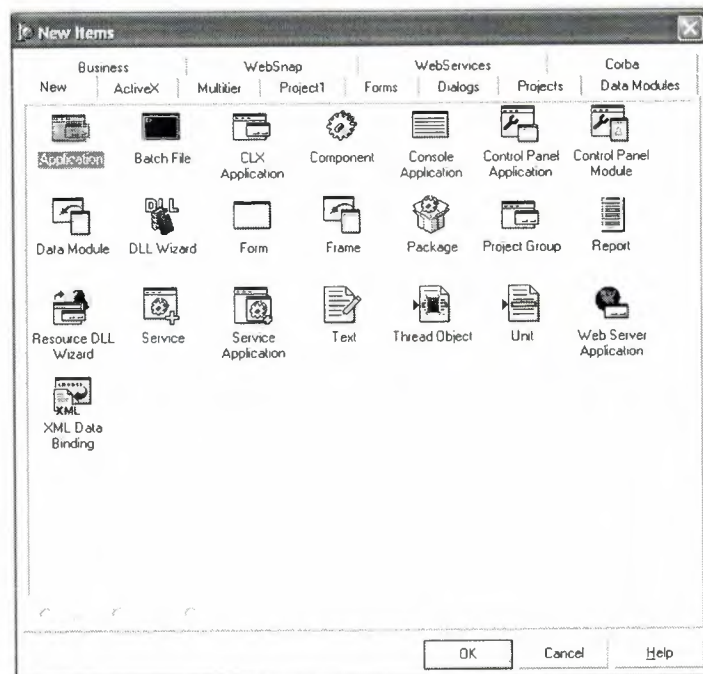


Figure 1.16 New Item

You can add your own objects to the Repository to facilitate reusing them and sharing them with other developers. Reusing objects lets you build families of applications with common user interfaces and functionality; building on an existing foundation also reduces development time and improves quality. The Object Repository provides a central location for tools that members of a development team can access over a network.

## **1.5 Programming with Delphi**

The following section provides an overview of software development with Delphi.

### **1.5.1 Starting a New Application**

Before beginning a new application, create a folder to hold the source files.

1. Create a folder in the Projects directory off the main Delphi directory.
2. Open a new project.

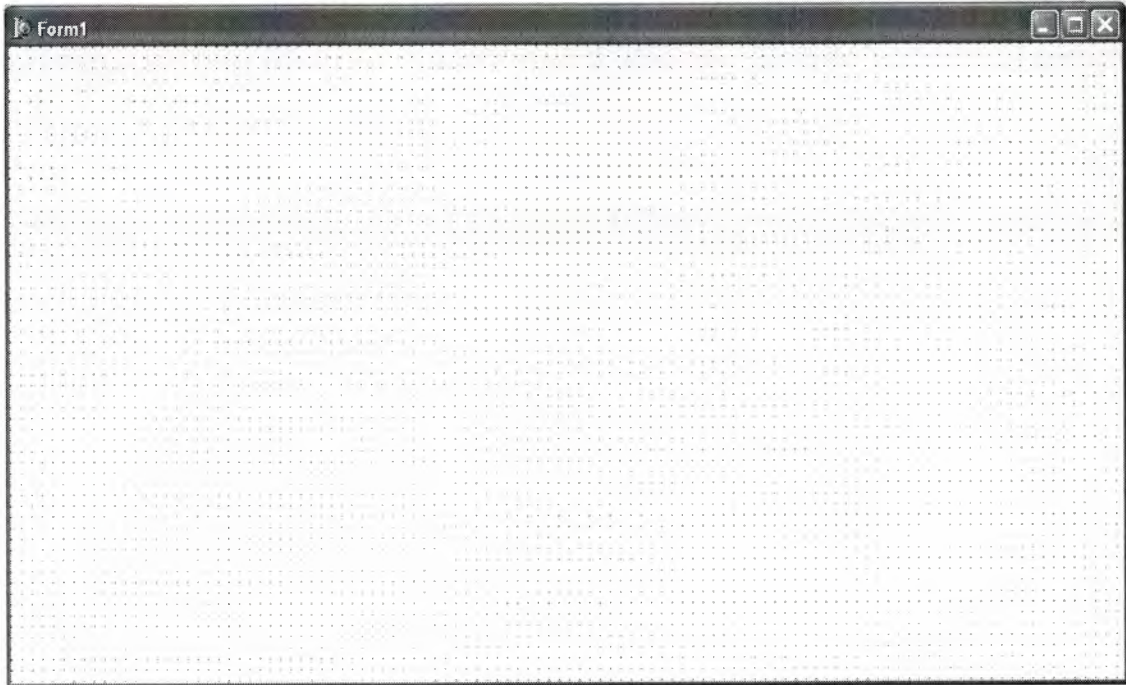
Each application is represented by a project. When you start Delphi, it opens a blank project by default. If another project is already open, choose File | New Application to create a new project. When you open a new project, Delphi automatically creates the following files.

- Project1.DPR : a source-code file associated with the project. This is called a project file.
  - Unit1.PAS : a source-code file associated with the main project form. This is called a unit file.
  - Unit1.DFM : a resource file that stores information about the main project form. This is called a form file.
3. Choose File | Save All to save your files to disk. When the Save dialog appears, navigate to your folder and save each file using its default name.

Later on, you can save your work at any time by choosing File | Save All.

When you save your project, Delphi creates additional files in your project directory. You don't need to worry about them but don't delete them.

When you open a new project, Delphi displays the project's main form, named Form1 by default. You'll create the user interface and other parts of your application by placing components on this form.



**Figure 1.17** Form Screen

The default form has maximize, minimize buttons and a close button, and a control menu

Next to the form, you'll see the Object Inspector, which you can use to set property values for the form and components you place on it.

The drop-down list at the top of the Object Inspector shows the current selected object. When an object is selected the Object Inspector shows its properties.

### **1.5.2 Setting Property Values**

When you use the Object Inspector to set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called design-time settings.

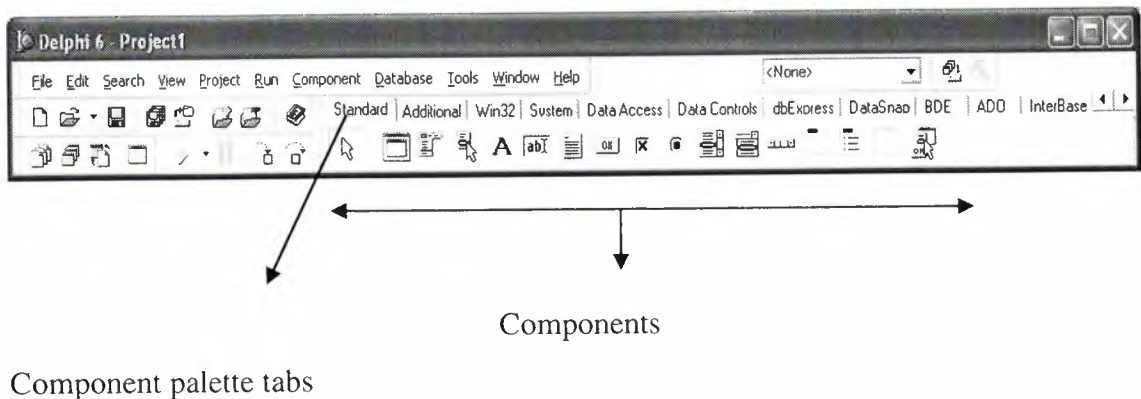
For Example; set the background color of Form1 to Aqua.



Find the form's Color property in the Object Inspector and click the drop-down list displayed to the right of the property. Choose clAqua from the list.

### 1.5.3 Adding objects to the form

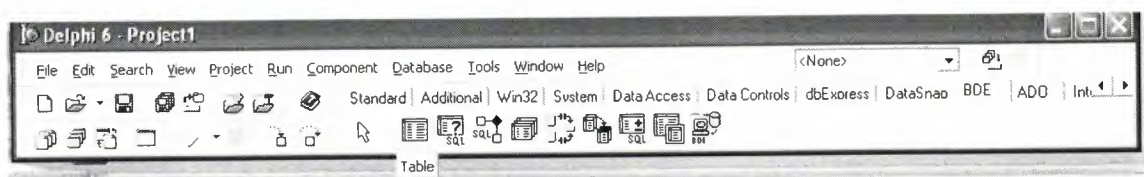
The Component palette represents components by icons grouped onto tabbed pages. Add a component to a form by selecting the component on the palette, then clicking on the form where you want to place it. You can also double-click a component to place it in the middle of the form.



**Figure 1.18** Standard Bar

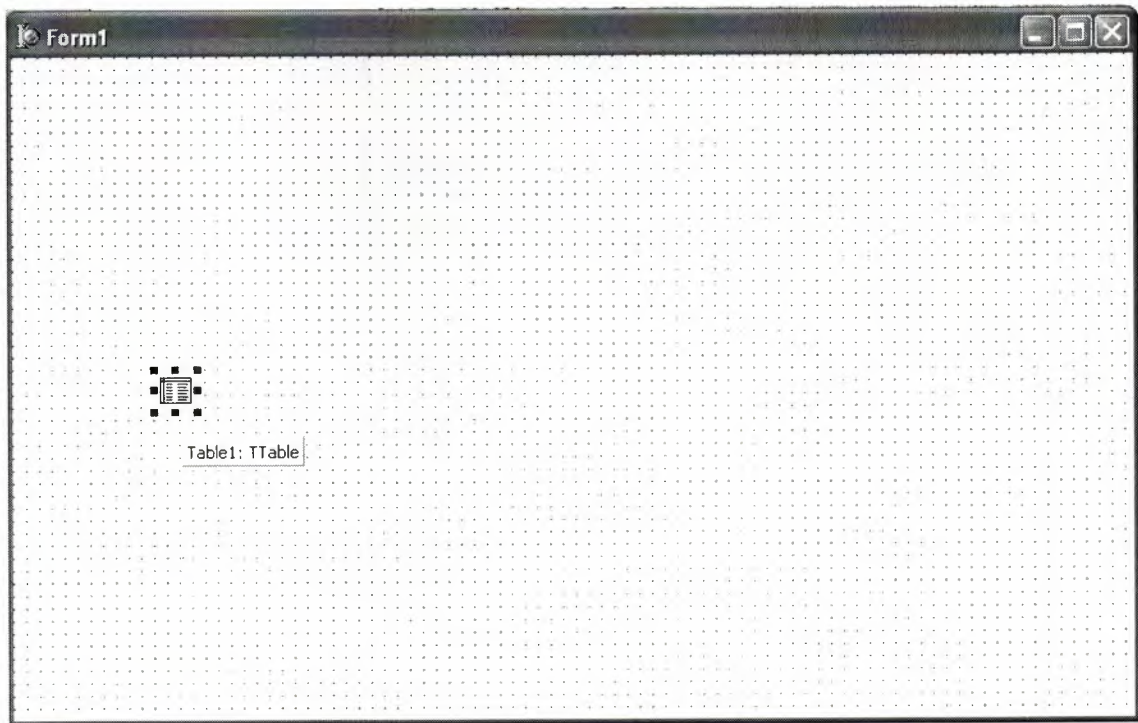
### 1.5.4 Add a Table and a StatusBar to the Form

Drop a Table component onto the form. Click the BDE tab on the Component palette. To find the Table component, point at an icon on the palette for a moment; Delphi displays a Help hint showing the name of the component.



**Figure 1.19** BDE Component palette

When you find the Table component, click it once to select it, and then click on the form to place the component. The Table component is non visual, so it doesn't matter where you put it. Delphi names the object Table1 by default. (When you point to the component on the form, Delphi displays its name—Table1—and the type of object it is—Table.)

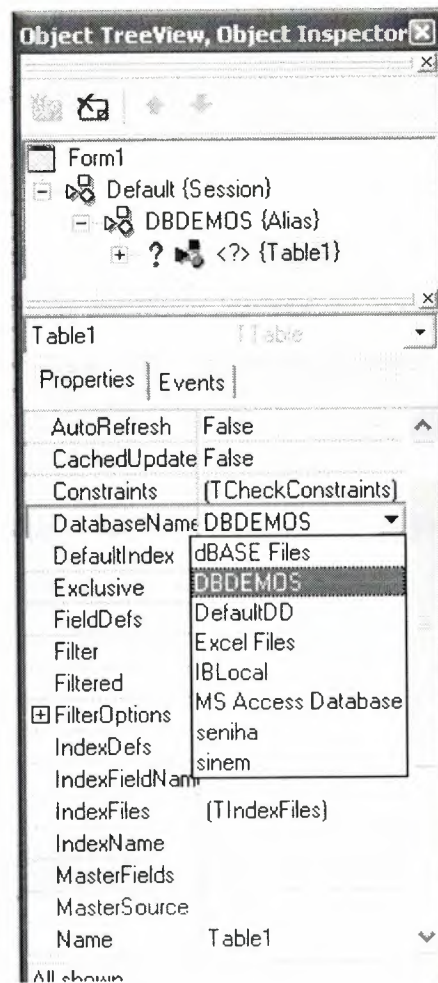


**Figure 1.20** Table in the Form

Each Delphi component is a class; placing a component on a form creates an instance of that class. Once the component is on the form, Delphi generates the code necessary to construct an instance object when your application is running.

Set the DatabaseName property of Table1 to DBDEMOS. (DBDEMOS is an alias to the sample database that you're going to use.)

Select Table1 on the form, and then choose the DatabaseName property in the Object Inspector. Select DBDEMOS from the drop-down list.



**Figure 1.21** Select DatabaseName

Double-click the StatusBar component on the Win32 page of the Component palette. This adds a status bar to the bottom of the application.

Set the AutoHint property of the status bar to True. The easiest way to do this is to double-click on False next to AutoHint in the Object Inspector. (Setting AutoHint to True allows Help hints to appear in the status bar at runtime.)

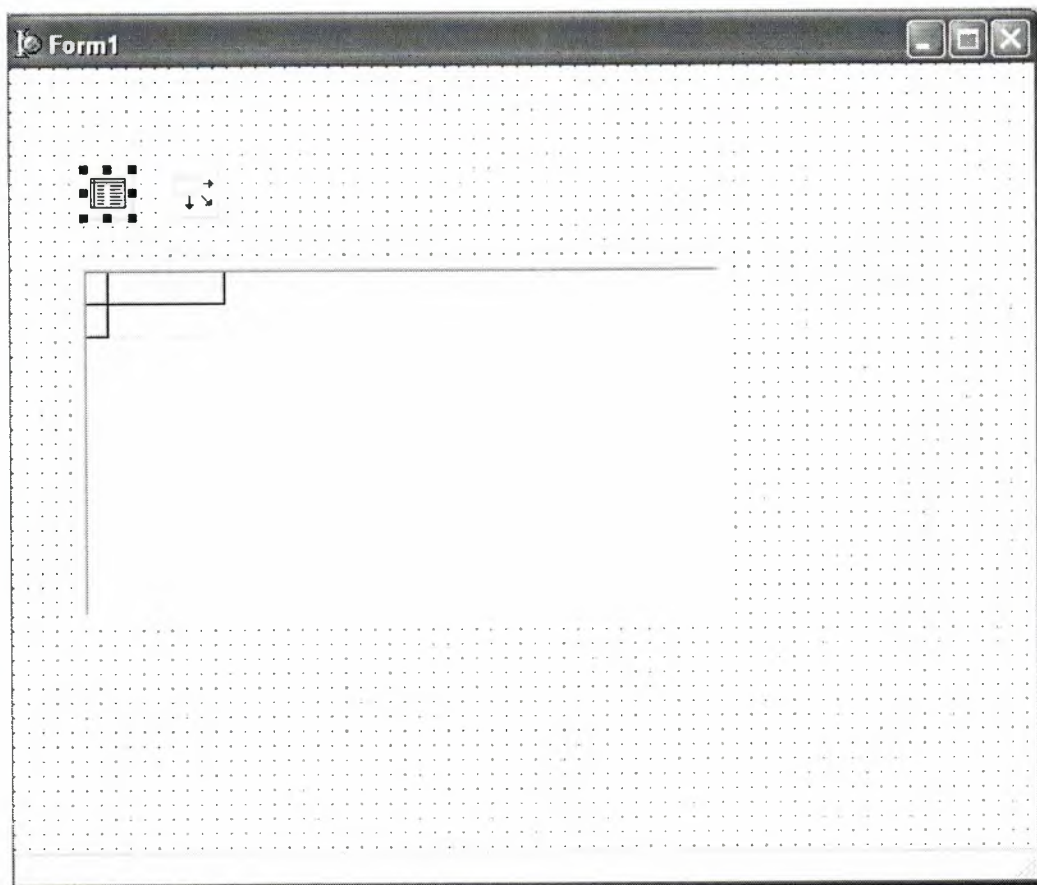
### 1.5.5 Connecting to a Database

The next step is to add database controls and a DataSource to your form.

1. From the Data Access page of the Component palette, drop a DataSource component onto the form. The DataSource component is non visual, so it doesn't matter where you put it on the form. Set its DataSet property to Table1.
2. From the Data Controls page, choose the DBGrid component and drop it onto your form. Position it in the lower left corner of the form above the status bar, and then expand it by dragging its upper right corner.

If necessary, you can enlarge the form by dragging its lower right corner. Your form should now resemble the following figure:

The Data Control page on Component palette holds components that let you view database tables.



**Figure 1.22** DBGrid in the Form

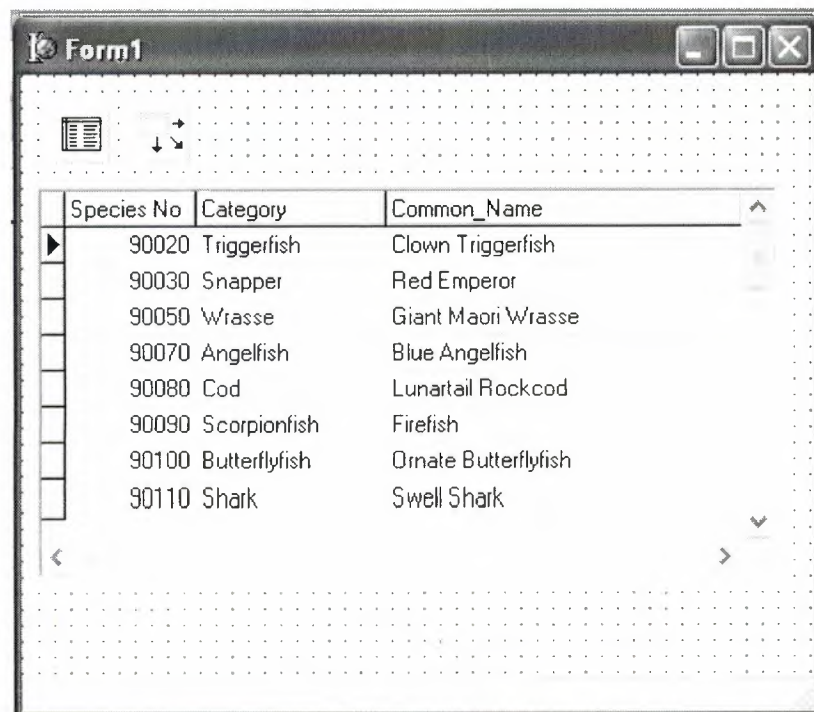


3. Set DBGrid properties to align the grid with the form. Double-click Anchors in the Object Inspector to display `akLeft`, `akTop`, `akRight`, and `akBottom`; set them all to true.
4. Set the `DataSource` property of DBGrid to `DataSource1` (the default name of the `DataSource` component you just added to the form).

Now you can finish setting up the `Table1` object you placed on the form earlier.

5. Select the `Table1` object on the form, and then set its `TableName` property to `BIOLIFE.DB`. (Name is still `Table1`.) Next, set the `Active` property to `True`.

When you set `Active` to `True`, the grid fills with data from the `BIOLIFE.DB` database table. If the grid doesn't display data, make sure you've correctly set the properties of all the objects on the form, as explained in the instructions above. (Also verify that you copied the sample database files into your `...\Borland Shared\Data` directory when you installed Delphi.)

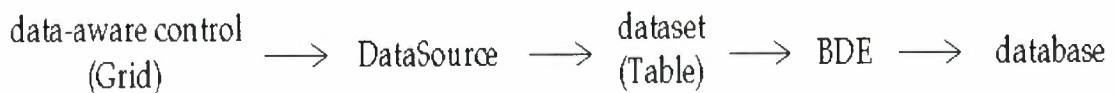


Species No	Category	Common_Name
90020	Triggerfish	Clown Triggerfish
90030	Snapper	Red Emperor
90050	Wrasse	Giant Maori Wrasse
90070	Angelfish	Blue Angelfish
90080	Cod	Lunartail Rockcod
90090	Scorpionfish	Firefish
90100	Butterflyfish	Ornate Butterflyfish
90110	Shark	Swell Shark

**Figure 1.23** Show Table

The DBGrid control displays data at design time, while you are working in the IDE. This allows you to verify that you've connected to the database correctly. You cannot, however, edit the data at design time; to edit the data in the table, you'll have to run the application.

6. Press F9 to compile and run the project. (You can also run the project by clicking the Run button on the Debug toolbar, or by choosing Run from the Run menu.)
7. In connecting our application to a database, we've used three components and several levels of indirection. A data-aware control (in this case, a DBGrid) points to a DataSource object, which in turn points to a dataset object (in this case, a Table). Finally, the dataset (Table1) points to an actual database table (BIOLIFE), which is accessed through the BDE alias DBDEMOS. (BDE aliases are configured through the BDE Administrator.)



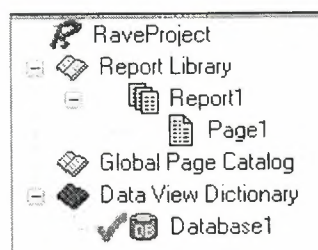
This architecture may seem complicated at first, but in the long run it simplifies development and maintenance. For more information, see “Developing database applications” in the Developer’s Guide or online Help.

## CHAPTER TWO

### 2 THE RAVE REPORTING

#### 2.1 Project Tree

The Project Tree provides an efficient way to visually manage all of the reports in your project. It quickly tells you the structure of your reporting project and the types of components contained on each page with icons that are the same as the component buttons. The Project Tree also visually shows parent-child relationships, the print order of component as well as the current selection (green check marks). You can select components by clicking on the component on the Page in the Visual Designer or on the Project Tree. Non-visual components appear only in the Project Tree in order not to clutter up your report design.



**Figure 2.1** Project Tree

There are three main sections in the Project Tree:

- The Report Library
- The Global Page Catalog
- The Data View Dictionary

Reports themselves can contain any number of page definitions. Global Pages are used to hold items that you want accessible to multiple reports. Data Views contain your field definitions and provide a link to the data in your application.

## 2.2 Design Tools

Rave is all about easy management. Besides making reporting easy and organized, Rave likes to keep itself organized and all according to what you want.

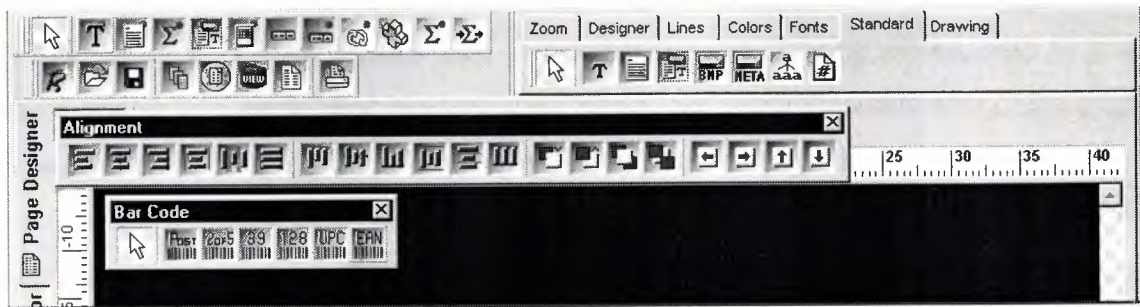


Figure 2.2 Toolbars

Since Rave is designed to be of ease to you there are three easy three ways for you to manage the many toolbars within Rave, which are:

- Tab-docking
- Normal docking
- Free-floating

Rave's many toolbars make it easy to design even the most complicated report. The toolbars include: Project, Designer, Zoom, Alignment, Color, Line, Font, Standard, Drawing, Report and Barcode component toolbars. Since it is possible to create and install new components, you may have other component toolbar buttons in your designer.



Figure 2.3 Project Toolbar

The Project toolbar provides quick access to project level functions such as New Project, Project Open, Project Save, New Report, New Global Page, New Data View, New Report Page or Execute Report.





**Figure 2.4** Designer Toolbar

The Designer toolbar allows you to change the characteristics of the Page in the Visual Designer. Characteristics such as whether the grid is being shown, snap to grid, draw grid on top, show band headers, show rulers, and show the waste area of the page. The last button brings up Rave's extensive Preferences dialog, which is described later.



**Figure 2.5** Zoom Toolbar

When you are working on a report with a complex design, you will find it much easier if you become familiar with the Zoom toolbar, which gives you quick access to Rave's extensive zooming capabilities. Select the zoom percent from a drop down list, type it in or use the Zoom Tool, Zoom In, Zoom Out, Zoom Selected, Zoom Page Width or Zoom Whole Page buttons.



**Figure 2.6** Alignment Toolbar

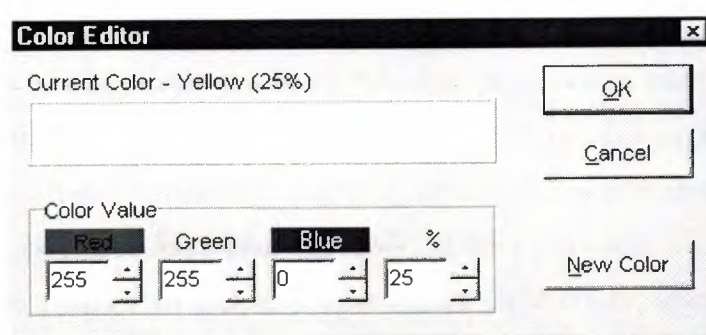
To help keep your report looking professional, Rave's Alignment toolbar provides access to a whole host of options to micro-manage the components on your page. The Left/Top, Center, Right/Bottom, Center In Parent, Space Equally, Equate Widths/Heights options offer the traditional alignment options. The Move Forward, Move Behind, Bring to Front and Send to Back order movement buttons allow you to change the print order of components and are visually backed up by the listing of the components in the Project Tree. Lastly, the buttons Tap Left, Tap Right, Tap Up and

Tap Down allow you to micro-adjust the position of components to the exact position you need.



**Figure 2.7** Colors Toolbar

The Color toolbar allows you to quickly select the primary and secondary colors of your components. There are 8 color spots that you can use to store any custom colors that you will be reusing throughout the project. If the colors available aren't enough, you can double click on the custom color palettes and create a different color using Rave's Color Editor (shown at right). With the Color Editor, you can select from a wider variety of colors or create your own combination of Red, Green and Blue and even select a percent saturation for the current color.



**Figure 2.8** Colors Editor

The Line toolbar is a useful tool for changing the line/border thickness and style for components such as Line and Circle. Sizes are listed in points instead of pixels so that your lines will always be the same thickness on your reports no matter the resolution of the printer that you are using.



**Figure 2.9** Line Toolbar

The Font toolbar provides quick access to a text component's font and alignment properties. It can also be useful for quickly viewing the font options for the currently selected text component(s).



Figure 2.10 Fonts Toolbar

## 2.3 Reuse and Maintenance Tools

Reports often take a large part of the development time for an application. Many times, there are many similarities between the design of separate reports.

This is where Rave's Mirroring technology comes in. When a component is set to mirror another, it assumes the appearance and properties of the component it is mirroring. The two components can be on the same page, across pages within the same report or on a global page. This is the primary purpose of a global page. You can almost think of it like an Object Repository, a central location for you to store reporting items that you want accessible to more than one report. If the component is a container control like TraveSection (similar to Delphi's Tpanel), all child components are mirrored as well. When the original component changes, all mirroring components will also change. While the mirrored component cannot change its properties, you can add additional components if it is a container control.

Here are just a few examples of where Mirroring would be useful:

Your customer wants a standard page header and footer on every page of their 50 reports. Now imagine you have all the reports done and your customer wants to change the layout of the headers and footers.

The Old Way – You would need to open up all 50 report definitions and change them one at a time.



The Rave Way – You would mirror the standard header and footer on each report you create and then any changes would only have to be done in one location. Also, if the standard header included a large bitmap, your reporting project would only contain a single copy rather than the many copies that a traditional report designer would require. You have to replicate a pre-printed form. The problem is there are 6 different variations of this form with only minor differences between each.

The Old Way – Assuming a traditional report designer could even handle this type of report, you would create the first form, cut and paste it into the second, make the minor modifications, then repeat for the other 4 forms, ending up with 6 reports that would be hard to maintain and take up a lot more memory.

The Rave Way – You would first create the common items of the form on a separate page, then mirror those on each form and add the unique parts for each as needed. If anything ever needed to be changed in the common section of the form, you would only need to change it in one place and since you're sharing most of the form's content, the report definitions take up much less room.

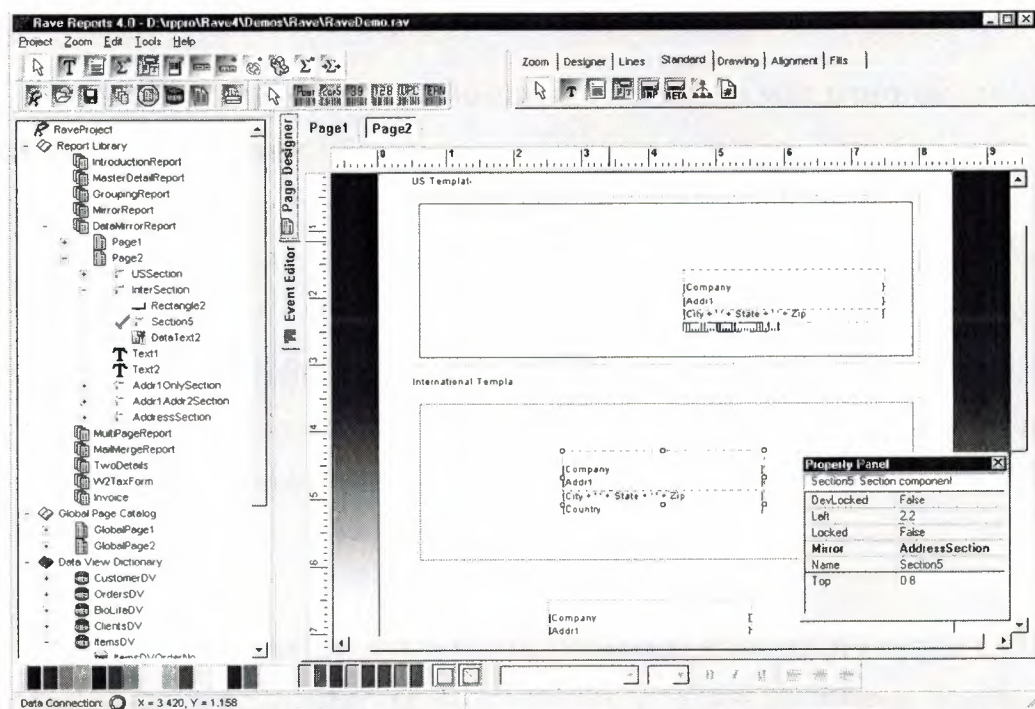


Figure 2.11 Mirror Report Example



Every text component has a FontMirror property which you can assign to a FontMaster component. This will allow you to change the fonts of many text controls from a single location. Imagine having Header, Body and Footer FontMaster components on a global page and changing the appearance of all of your reports with just a few mouse clicks.

Another important aspect of maintaining any large project is documentation. The Project and every Report, Page, Data View and Data Field component has a multi-line Description Property that can be used to comment the intended usage or other information. This can be useful if you are coming back to a project that you last worked on 6 months ago or especially if another programmer or your end user will be modifying reports that you created.

## 2.4 Standard Components



**Figure 2.12** Standard Tool Bar

**Text** – This component is used to display fixed text on your report for items such as column headers or report titles.

**Memo** – This component is used to display fixed text in a word wrapped fashion on your report. Using the MailMergeItems property and the Mail Merge Editor shown below, you can create a mail merge type of report where Rave will replace tokens in the memo text with a replacement string. Note that this replacement string can be edited with the Edit button, which will display the Data Text Editor for quite a bit of extra functionality.

**Section** – This component is a terrific component manager. It acts as a container for other components, in other words it help you to group components together. By

properly using section components and mirroring, you can create reusable and maintainable reports in no time flat.

**Bitmap** – This component is used to display a bitmap (\*.bmp). Through the FileLink property you can reference a file on the hard disk.

**MetaFile** – This component is used to display a metafile (\*.wmf). Through the FileLink property you can reference a file on the hard disk.

**FontMaster** – This component is used to control the font characteristics of any text control through their FontMirror properties. See Reuse and Maintenance for more information.

## 2.5 Drawing Components

**Line** – Draws a diagonal line. (This may not seem like a unique feature but did you know that most Delphi reporting tools cannot create a diagonal line visually.)



**Figure 2.13** Drawing Tool Bar

**Hline** – Draws a horizontal line.

**Vline** – Draws a vertical line.

**Rectangle** – Draws a rectangle.

**Square** – Draws a square.

**Ellipse** – Draws an ellipse.

**Circle** – Draws a circle.

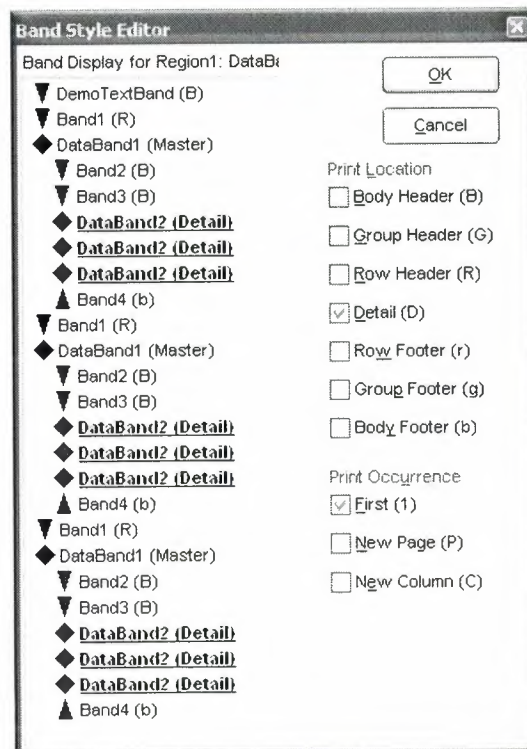
## 2.6 Reporting Components

**Region** – This component acts as a container for Band and DataBand components. To create a composite or sub-report, simply drop more than one region on a page and add the appropriate bands to each.



**Figure 2.14** Report Tool Bar

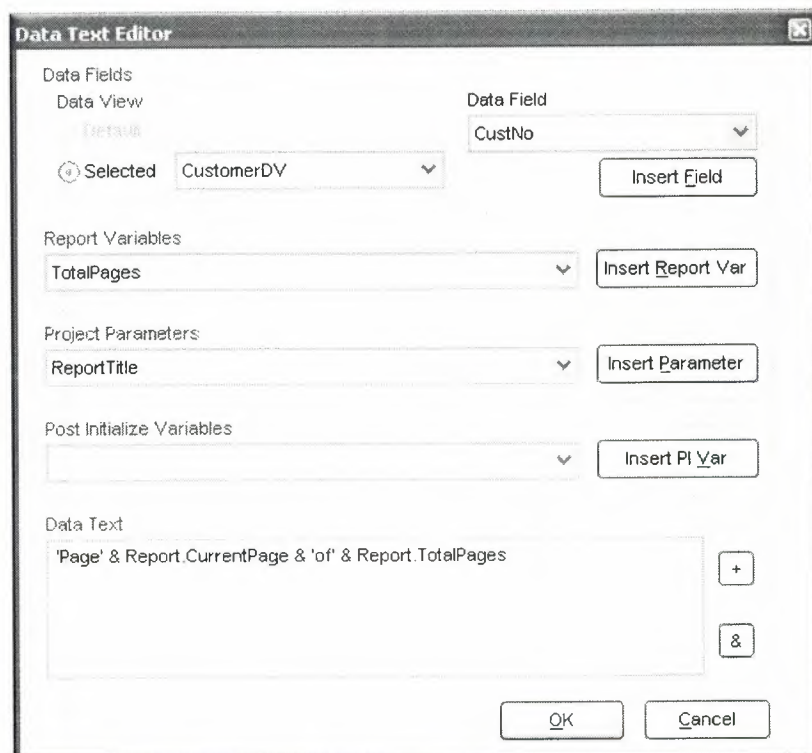
**Band** – This component is primarily used to create header and footer bands in a banded style report. A Band component can only be created within a region and its purpose is controlled through the Band Style Editor shown below. The Band Style Editor displays a virtual layout of all of your bands for the given print locations of each band or data band. Note that you can create as many Bands as you like and a Band may print in multiple locations if the report design requires it. So for example, if you want a solid horizontal line to appear above and below a detail body, you could create a single band and set it to print on both the Body Header and Body Footer. You can also control the Print Occurrence for a Band, having it continue on a new page or column or any combination of occurrence settings. You can set a Band to group on specific fields and can create as many different types of group headers or footers as your report requires. Basically, with Rave's Band and DataBand components, you'll be able to create just about any banded style layout that you can imagine.



**Figure 2.15** Band Style Editor

**DataBand** – The DataBand component is fairly similar to a band component except that it is tied to a particular DataView and iterates across the rows in the DataView. You can link DataBands together for Master-Detail to unlimited levels or multiple details on the same level. Some advanced features that are supported by a DataBand include KeepBodyTogether, KeepRowTogether, StartNewPage, MaxRows and Orphan/Widow control.

**DataText** – The DataText component is the primary means to output fields from your database. You can quickly select a specific DataView and DataField with Property Panel or use the Data Text Editor shown below to create any combination of string constants, data fields, report variables or project parameters. The & concatenation operator is the same as the + operator, except that it also inserts a space. Report Variables are items such as total pages or current date and time in a variety of formats. Project Parameters are custom variables that you create and initialize from your Delphi application. Project Parameters can be used for items such as user defined report titles, printing the current user name or other custom information.



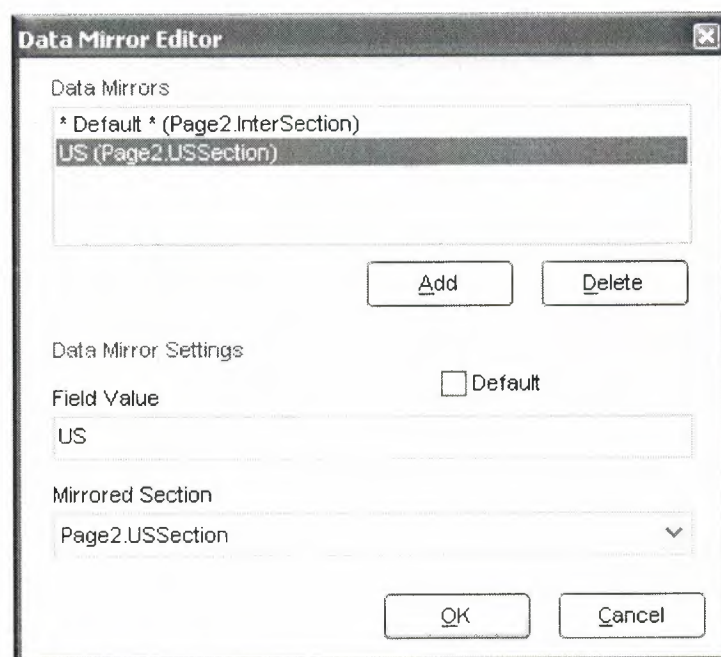
**Figure 2.16** Data Text Editor



**DataMemo** – This component is very similar to the Memo component except that it retrieves data from a DataField. DataMemo component's print text data out in a word wrapped fashion and the DataField can be any text type, not just memo fields. It also has RTF and mail merge support.

**CalcText** – This component is used to perform simple operations such as Sum, Average, Count, Min and Max on a data field. You can set the value as a running total and place it in any type of band or anywhere on the page) you need it.

**DataMirrorSection** – The data mirror section component is similar to Rave's section component (found in the Standard Toolbar) with one major difference, it will dynamically mirror another section depending upon the value of a DataField. You configure the data mirror section using the Data Mirror Editor (shown below). This component is very useful for printing out data that has different formats depending upon the type of data. One example is an address field that could print a US format if the country field is "US" and an international format otherwise (using the Default option in the Data Mirror Editor). You could also print Boolean field values with your own custom bitmaps.



**Figure 2.17** Data Mirror Editor

## 2.7 Barcode Components



Figure 2.18 Barcode Toolbar

PostNetBarCode – Prints a US PostNet bar code.

I2of5BarCode – Prints Interleaved 2 of 5 barcodes.

Code39BarCode – Prints standard and extended Code 39 barcodes.

Code128BarCode – Prints A, B and C Code 128 barcodes.

UPCBarCode – Prints UPC-12 barcodes.

EANBarCode – Prints EAN-13 barcodes.

## 2.8 Anchors

Anchors are a powerful way to create a report that dynamically adjusts to changing sizes. This allows you to create reports that can print well whether the user selects landscape or portrait, 8.5" by 11" or A4. There are 6 different anchor values for both the horizontal and vertical dimensions to allow you to control each component in exactly the manner that it needs. The Anchor Editor (shown at right) even shows you a helpful bitmap of how each anchor setting works.

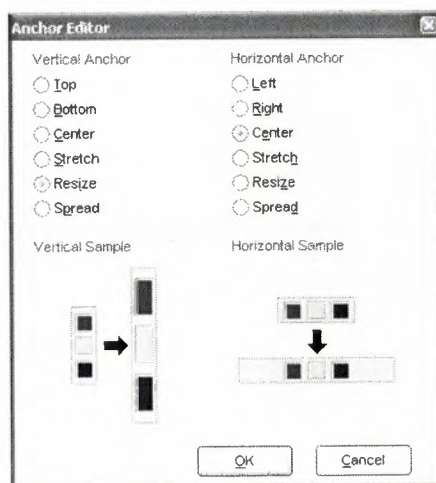


Figure 2.19 Anchor Editor

## 2.9 Code Based Reports

Lately Delphi has decided to include Rave Reports as the default reporting solution, replacing Quick Reports. Since they work in very different paradigms, many people were confused by the new environment. This is intended as an introduction for people who haven't worked with Rave yet, and would like to start.

Nowadays Delphi ships with Rave Reports 5.0.8. If you haven't already, download the update from the registered users page, since it fixes some important problems.

You can develop reports with Rave using two different ways: Code Based or with the Visual Designer.

With Code Based, you write reports using plain Delphi code. That provides a very flexible way displaying any kind of data, allowing any kind of complex layouts.

To write a code based report, just drop a TrvSystem component on the form and write the report on the OnPrint event handler. Sender is the report you are creating, and can be typecasted to TbaseReport. It contains all the methods you need to output information to that particular report.

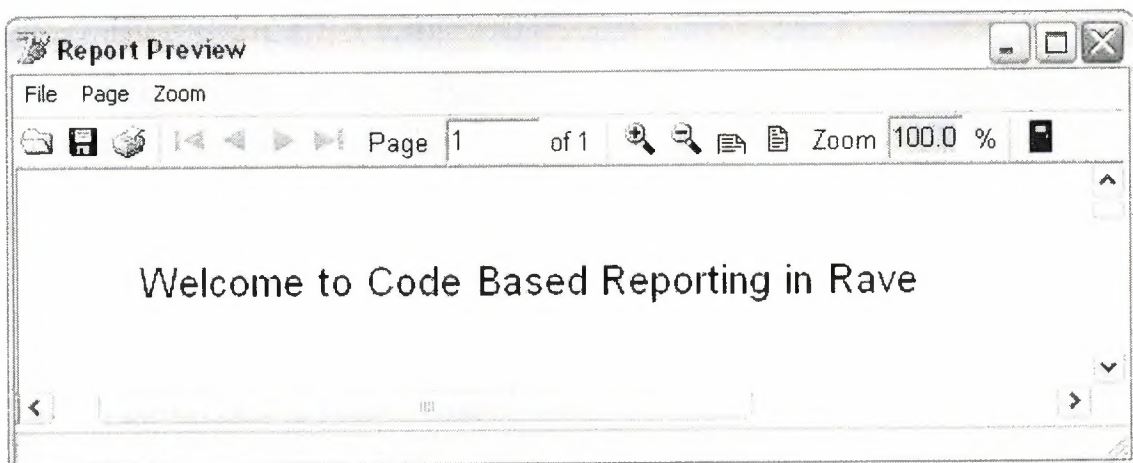
### 2.9.1 Simple Code Base Report

Here's a simple report using the code based mechanism:

```
procedure TFormMain.RvSystemPrint(Sender: Tobject);
begin
  with Sender as TbaseReport do
  begin
    SetFont('Arial', 15);
    GotoXY(1,1);
    Print('Welcome to Code Based Reporting in Rave');
  end;
end;
```

To execute this report, call `RvSystem.Execute` method.

So, what does that simple code do? First, it calls `SetFont` to select the font and size of the text that will be printed from that point on. Then it positions the cursor on the coordinates (1,1). These coordinates are expressed using the units set in the `SystemPrinter.Units` property of the `RvSystem` object, and it defaults to Inches. You can set it to `unUser` and set a number relative to Inches in the `SystemPrinter.UnitsFactor` property. For example, if `UnitsFactor` was set to 0.5 then 1 unit would correspond to half an inch. Finally, the code calls the `Print` method to output the text. Here's the output:



**Figure 2.20** Report Preview

### 2.9.2 Tabular Code Based Report

Here's another example. It displays a list of the folders in the root of the current drive, along with a recursive count of number of files and folder, and total size of the files included in each folder.

```
Procedure TFormMain.PrintTabularReport(Report: TbaseReport);  
var  
    FolderList : TstringList;  
    I          : Integer;  
    NumFiles   : Cardinal;
```



```

NumFolders : Cardinal;
SizeFiles : Cardinal;
Root      : string;
begin
  with Report do
  begin
    SetFont('Arial', 15);
    NewLine;
    PrintCenter('List of Folders in the Drive Root', 4);
    NewLine;
    NewLine;
    ClearTabs;
    SetTab(0.2, pjLeft, 1.7, 0, 0, 0);
    SetTab(1.7, pjRight, 3.1, 0, 0, 0);
    SetTab(3.1, pjRight, 3.5, 0, 0, 0);
    SetTab(3.5, pjRight, 4.5, 0, 0, 0);
    SetFont('Arial', 10);
    Bold := True;
    PrintTab('Folder Name');
    PrintTab('Number of Files');
    PrintTab('Number of Folders');
    PrintTab('Size of Files');
    Bold := False;
    NewLine;
    FolderList := TStringList.Create;
  try
    Root := IncludeTrailingPathDelimiter(ExtractFileDrive(ParamStr(0)));
    EnumFolders(FolderList, Root);
    for I := 0 to FolderList.Count - 1 do
    begin
      PrintTab(FolderList[I]);
      GetFolderInfo(IncludeTrailingPathDelimiter(Root+FolderList[I]),
        NumFiles, NumFolders, SizeFiles);
    end
  end
end

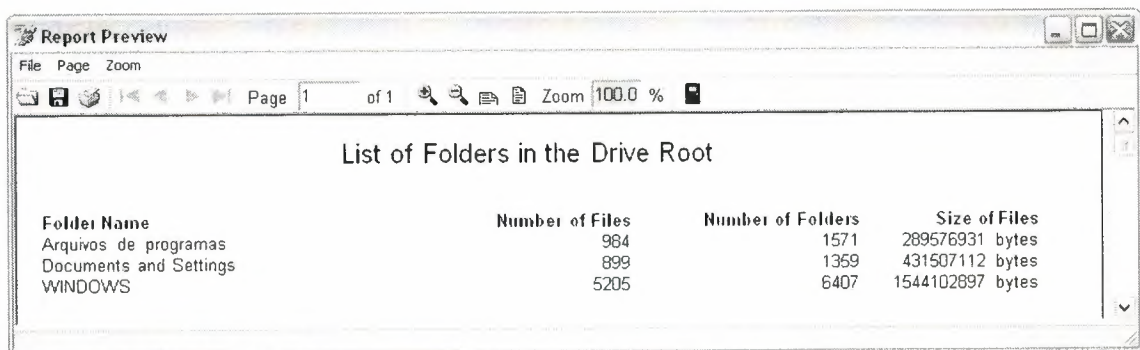
```

```

PrintTab(Format('%u',[NumFiles]));
PrintTab(Format('%u',[NumFolders]));
PrintTab(Format('%u bytes',[SizeFiles]));
NewLine;
end;
finally
  FolderList.Free;
end;
end;
end;
end;

```

Notice that a different approach has been taken: instead of specifying the coordinates of each text output, the printing was done using Lines and Columns as references. The line height depends on the size of the current font: each unit represents 1/72nds of an inch, so each line printed with a size 10 font will have, appropriately, a height of 0.138 inches. Lines are advanced after calls to PrintLn or NewLine. Columns are defined using calls to the SetTabs method, and the PrintTab method will print the text in the current column and advance to the next one. Here's the output:



The screenshot shows a window titled 'Report Preview' with a menu bar (File, Page, Zoom) and a toolbar. The main content area displays a table titled 'List of Folders in the Drive Root'. The table has four columns: 'Folder Name', 'Number of Files', 'Number of Folders', and 'Size of Files'. The data rows are: 'Arquivos de programas' (984 files, 1571 folders, 289576931 bytes), 'Documents and Settings' (899 files, 1359 folders, 431507112 bytes), and 'WINDOWS' (5205 files, 6407 folders, 1544102897 bytes).

Folder Name	Number of Files	Number of Folders	Size of Files
Arquivos de programas	984	1571	289576931 bytes
Documents and Settings	899	1359	431507112 bytes
WINDOWS	5205	6407	1544102897 bytes

**Figure 2.21** Report Preview

### 2.9.3 Graphical Code Based Report

You can include shapes and images in your code based report, along with the text. The following example demonstrates that:

```
procedure TFormMain.PrintGraphicsReport(Report: TbaseReport);
```

```
var
```

```

    Bitmap : Tbitmap;
begin
    with Report do
    begin
        Canvas.Brush.Color := clGray;
        Rectangle(0.3, 0.3, 4.7, 3.3);
        SetFont('Arial', 15);
        FontColor := clRed;
        PrintXY(0.5,0.5, 'Just look at all the graphics!');
        Bitmap := Tbitmap.Create;
    try
        Bitmap.LoadFromFile('delphi.bmp');
        PrintBitmap(3.5,0.3,1,1, Bitmap);
        PrintBitmap(1,2,3,3, Bitmap);
        Canvas.Pen.Color := clBlue;
        Canvas.Brush.Bitmap := Bitmap;
        Ellipse(5,0.3,6,3.3);
        Ellipse(2,1,4,1.9);
    finally
        Bitmap.Free;
    end;
    Canvas.Pen.Color := clBlack;
    Canvas.Brush.Style := bsSolid;
    Canvas.Brush.Color := clYellow;
    Pie(0.7,0.7,1.7,1.7,1,1,1,2);
    Canvas.Brush.Color := clGreen;
    Pie(0.7,0.7,1.7,1.7,1,2,1,1);
    end;
end;

```

In this example the methods Rectangle, Ellipse and Pie have been used draw shapes with different fills. Bitmaps were outputted using PrintBitmap and as the brush of the ellipses. Here's the output:

## Graphics Report Example

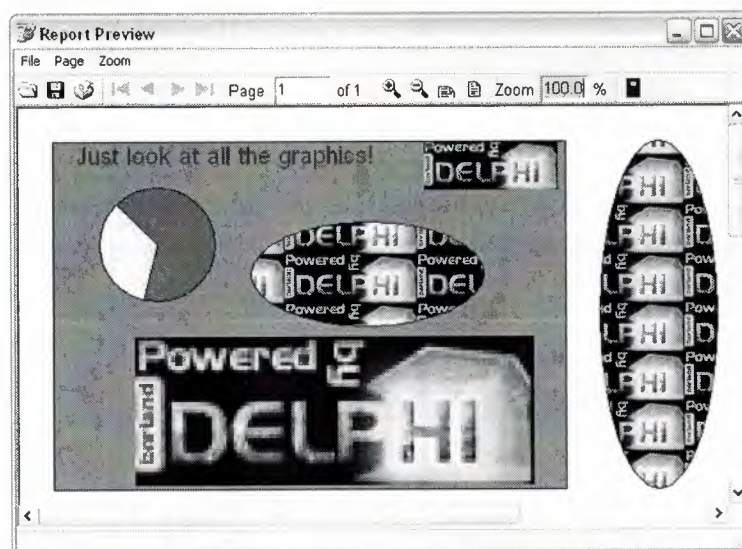


Figure 2.22 Report Preview

## 2.10 Visually Designed Reports

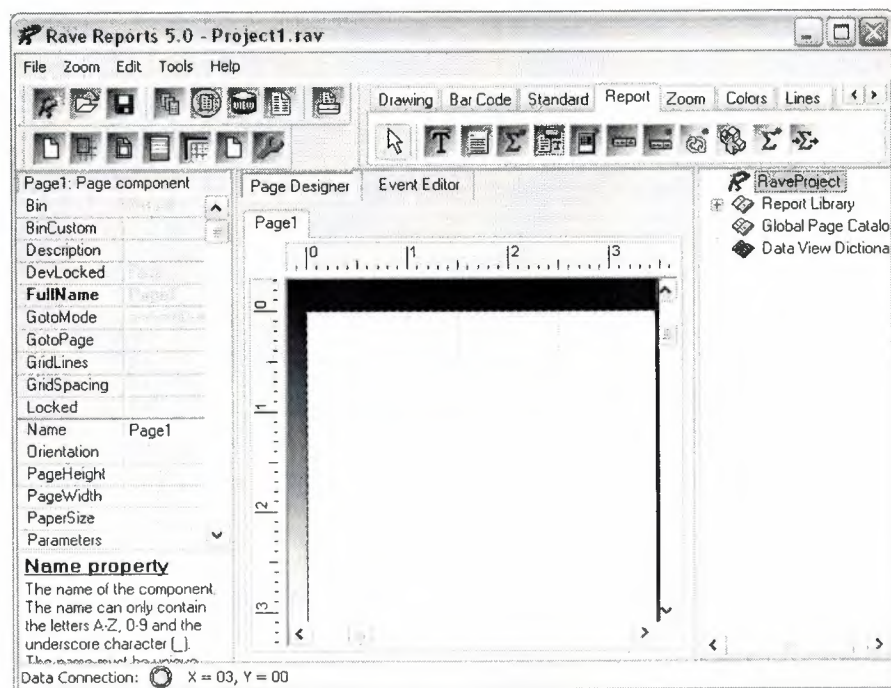
### 2.10.1 The Visual Designer

If you are used to work with Quick Reports, the default reporting engine included in the previous versions of Delphi, you created your reports using Delphi's own form designer, and they were save in the DFM, included as resources in your executable. Rave works a bit differently in this aspect: it has it's own report designer, and saves the report using it's own file format. This has some advantages, including the fact that your reports can be made "standalone", and be used or updated independently of your application, or even made available in a Intranet or in the Internet, using Nevrona's Rave Report Server. Of course, you can still have it saved in a form's DFM.

To get started with the Rave Visual Designer, drop a TrvProject in a form. This will be the link from your application to the reports you are developing. If you want, you can add a TrvSystem and link your RvProject to it, through it's Engine property. The RvSystem is the object responsible for the general configuration of the reports: the printer that is going to be used, the margins, the number of pages, and so on. To start a new project, double click the RvProject you added to the form, or select "Rave Visual Designer" from its context menu.




This is the interface that you will be working on:



**Figure 2.23** Rave Visual Designer

The interface is simple, and you might be familiar with some parts of it from Delphi's IDE. On the top there's the menu, the toolbar, and the component palette that contain the components that will be used in the reports. In the left there's the Object Inspector, which will be used to adjust the properties of the components of the report. In the middle there's the Page Designer or the Event Editor, and in the left there's the very usefull Project Treeview. For a quick overview of the components in the palette, you can go to Nevrona's Visual Designer page.

A Rave Project File can have one or more reports. That way you can keep common items between them in a single location, called Global Pages. If you expand the Report Library node of the Project Treeview, you can see that right now you are working on Report1. Clicking on it, its properties will show on the Inspector. Let's change it's name and call it SimpleReport. Next, go to the Standard tab on the Component Palette, and pick a Text component and add it to the page. Change its text property, and adjust its size and position. Here's how it looks like:

Anchor	(Top / Left)
Color	 Black
DevLocked	False
DisplayOn	doParent
<b>Font</b>	<b>Arial,15</b>
FontJustify	plLeft
FontMirror	
Left	1,000
Locked	False
Mirror	
Name	Text1
Rotation	0
Tag	0
<b>Text</b>	<b>Welcome to Rave Reports Visual I</b>
Top	1,000
<b>Truncate</b>	<b>True</b>
Width	4,000

**Figure 2.24** Component Palette: Standard Tab

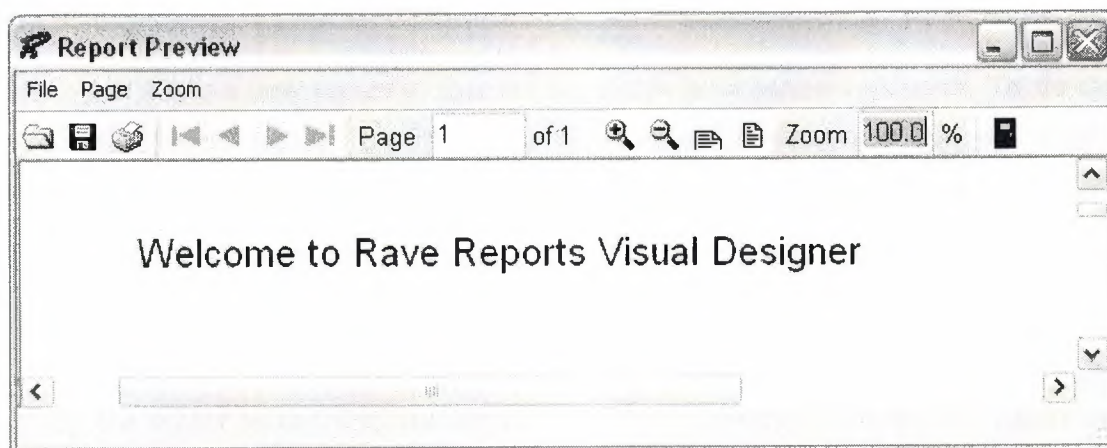
As you can see, the properties that were changed from the default values are shown in bold. In this case, I changed the Font, Text and Truncate properties. By default it does not highlight Name, Pos and Size changes. If you'd like to see them, right click the Inspector and uncheck "Exclude Name, Size and Pos changes" in the context menu.

You might have also noticed that Rave does not have an auto size property. You can use the Truncate property to have that effect: if truncate is false, the design time size will have no effect.

You can see the result of this simple report right on the designer: Press F9 or use File/Execute Report to run it. Now let's do it in our application. Save your project and return to Delphi. Change to ProjectFile property of RvProject to point to the file you just saved. To run the report, add a call to the Execute method of the RvProject object in a button click, for example.

RvProject.Execute will only work for now because we only have one report in this project. If we had multiple reports, we'd have to call SelectReport to choose one before calling Execute, or calling ExecuteReport directly.

Here's the output:



**Figure 2.25** Report Preview

Tip: If you Close and Open your project before executing, you won't need to recompile your application or restart it to see the changes you just made in the designer.

### **2.10.2 Interacting with the Project**

If you worked with Quick Reports, you might be used to manipulating the objects in runtime, changing their Position, Text and Visibility. After all, they were just Tobjects! While this is possible with Rave, and I'll cover it in a later article, it's a little harder than it was with QR. But don't worry, Rave provides a different answer to this kind of problems.

#### **Parameters**

If you can use parameters in your reports. They can be defined using the parameters property of either the Project, a Report or a Page. Parameters can be defined in either of these places, they are just in multiple places for easier access.

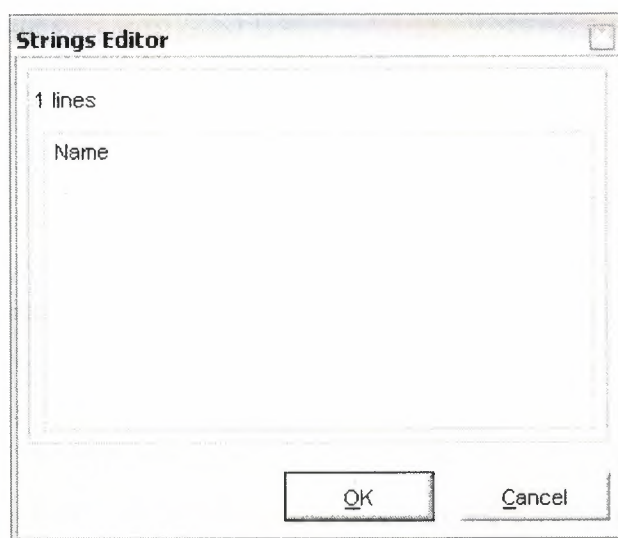
You can only select the Project and a Report through the Project Treeview. A page, however, can be selected using the Project Treeview or clicking on it's title above the page designer.



Among other uses, you can print parameters. So, for instance, if the title of your report can be user-defined, you could pass it from your application into the report as a parameter.

Let's add a new report to this project to see how parameters work. To do that, click the fourth button on the toolbar or choose File/New Report. Call it ParametrizedReport, changing its name through the object inspector. This report is going to be very similar to the first one, except the text is going to be user-defined.

Now we need to define the parameter that is going to be printed. To do that, still having the report as the selected object, open the property editor the the parameters property. There should be listed all parameters of this report, each on a separate line. Add a parameter called Name, like this:



**Figure 2.26** Strings Editor

Parameters can be printed using a DataText component, available in the Report tab of the component pallette. Add a DataText to the page, and open the property editor of the DataField property. There you can choose which field is going to be printed, when working with DataAware reports. You can also choose Project Variables, Parameters and Post-Initialize Variables from there.



So choose the parameter added previously from the Parameters drop-down combo and press the Insert Parameter button. The data text expression is now Param.Name. Press OK and try to execute the report, as before. Nothing is printed, since the parameter has not been set.

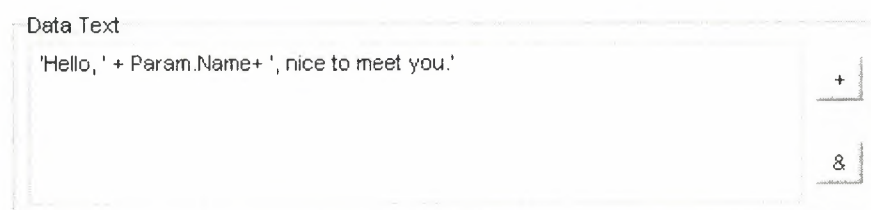
We need to set this parameter before printing. Don't forget to save your changes, and return to Delphi, adding a call to SelectReport before Execute, so we can see the right report. Before executing, though, we need to set the parameter we added. That is made using RvProject's SetParam method. This is how my code looks like right now:

```
procedure TFormMain.btnExecuteClick(Sender: TObject);
begin
    RvProject.Open;
    RvProject.SelectReport('ParametrizedReport',False);
    RvProject.SetParam('Name','Leonel');
    RvProject.Execute;
    RvProject.Close;
end;
```

Now, when we execute the report, we are going to see the string we set as a parameter printed.

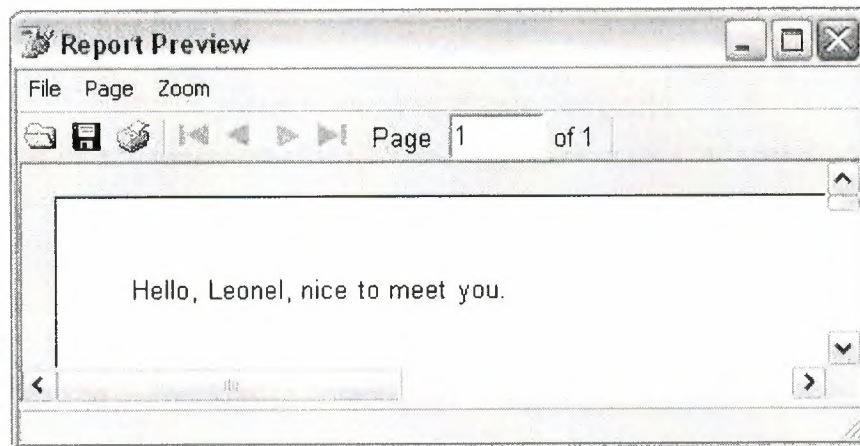
Tip: You can use RvProject.GetReportList to get a list of available projects, and add them to a ComboBox, or a RadioGroup, for example. That makes selecting the report easier.

But this is too simple. Let's change the expression that is going to be printed. Return to Rave Designer and open the property editor for the DataText we added. You can add any text you want, combining text, fields, parameters and variables. I changed it to this:



**Figure 2.27** Data Text Sample

Here's the result:



**Figure 2.28** Report Preview

### Post-Initialize Variables

Post-Initialize Variables, or simply PI Vars, are variables whose value is only known after the report has already been printed. It may sound strange, at first, but think about the number of pages of a report, for example. We can only know its value after the report is ready. Actually TotalPages is a report variable that acts like a PI var, and can easily be printed using DataTexts as we did with Parameters.

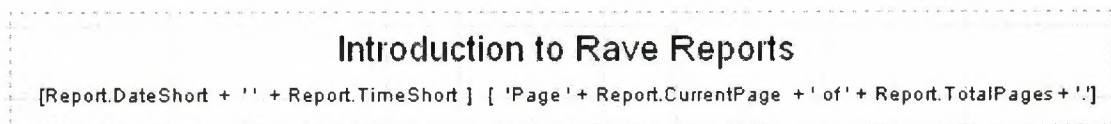
### Global Pages

When you have parts of reports that are common to two or more reports, you can put these in a global page. Let's suppose we have a header with our company name, the date and time that report is being printed, the current page and the number of pages of that report. We want that header to be in every report. How can we do it?

First, add a global page to the project, using File/New Global Page, or the Toolbar shortcut. In that page, add a section component, available in the standard tab of the component palette.

Sections are logical groupings of components. They can be used to group component so they can be easily moved around the report or as containers for Mirrors, as we are doing right now.

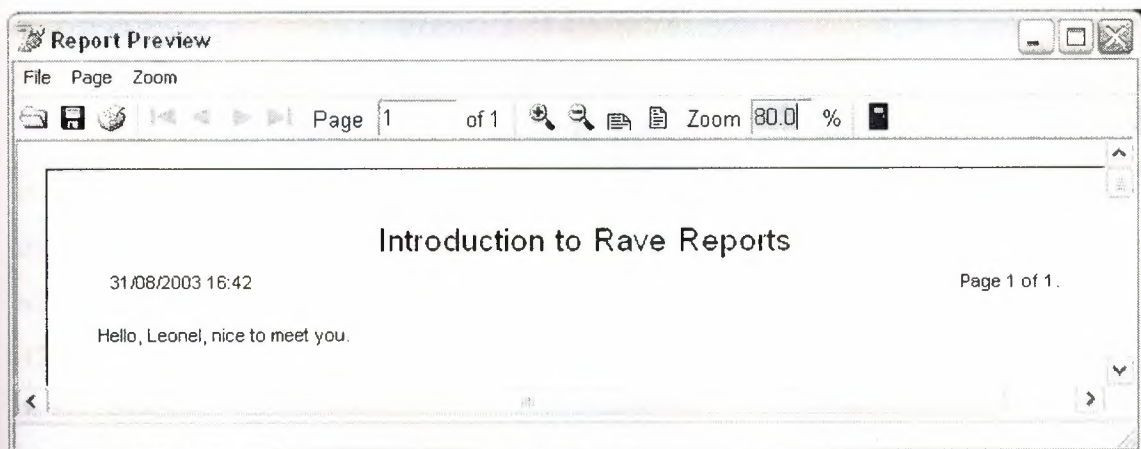
Inside that section we add what we want to be printed. In this case, a few DataTexts. My header looks like this:



**Figure 2.29** Header Sample

Hint: Instead of changing the font property of several components to the same font, link them to a FontMaster component, available in the standard tab, and set the font on it. That way is easier to change the font in the future, in case it's needed.

Now add another section to the Page1 of SimpleReport. Set its Mirror property to GlobalPage1.Section1. You will see a copy of the header you created in the global page. Do the same thing to ParametrizedReport. Now both reports share the same header. Here how it looks like:



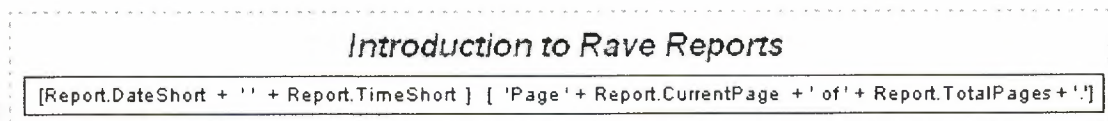
**Figure 2.30** Report Preview

### Conditional Printing

Sometimes we need to print certain parts of a reporting depending of some conditions. Rave has a very powerful way of dealing with this. We can conditionally mirror sections depending on field values or parameters. Let's create a new Report, calling it a ConditionalReport.

Let's pretend that this new report is a trick one. The user can choose the header that is going to be printed, from two different kinds of headers. He can also choose for the report to be printed without a header. We are going to use a parameter to tell the report what kind of header is going to be printed, and a DataMirrorSection to select the proper header at runtime.

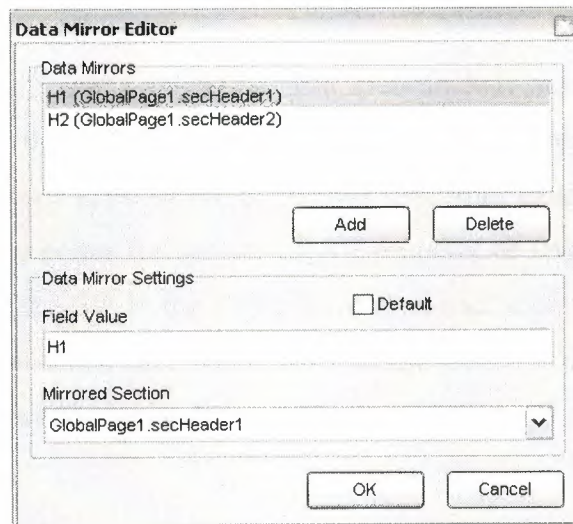
First, add a parameter to this new report called HeaderKind. Let's assume that it will have the values H0 (for no header), H1 (for the first header), H2 (for the second kind of header). Now add a new section to the global page (you can reach it through the Project Treeview), with the second kind of header layout. I created a header similar to the first one, changing the font title and adding a border around the values. It looks like this:



**Figure 2.31** Header Sample

Now return to the Page1 of ConditionalReport, and add a DataMirrorSection, available at the Report tab of the component pallette. Go to its DataField property editor, and set Param.HeaderKind as the expression. Now go to the DataMirrors property editor, and add two Data Mirrors: if the value is H1, it should point to the first header, H2, to the second. Since H0 does not match any mirrors, nothing will be printed. It should look like this:





**Figure 2.32** Data Mirror Editor

Notice that I gave more meaningful names to each of the sections earlier.

Hint: You can use the OnMirrorValue event of the DataMirrorSection to work on ranges of values.

Now return to Delphi and add the code to set the parameter according to the user's choice. I added a ComboBox with the options and my code looks like this:

```
Procedure TFormMain.btnExecuteClick(Sender: TObject);
```

```
Begin
```

```
  RvProject.Open;
```

```
  RvProject.SelectReport (cmbReports.Text, False);
```

```
  case cmbReports.ItemIndex of
```

```
    1: RvProject.SetParam('Name',edName.Text);
```

```
    2: RvProject.SetParam('HeaderKind',Format('H%d',[cmbHeaderKind.ItemIndex]));
```

```
  end;
```

```
  RvProject.Execute;
```

```
  RvProject.Close;
```

```
end;
```

Now the proper header will be printed according to the user's choice.

Embedding the Project in the Executable

When you deploy your application, you must include your project file. You can have it as a separated file, so you can update it in a easier way, only shipping a new one, without recompiling your application, or include it in your executable. It's easy to do that: open the property editor for the StoreRAV property of RvProject. There you can press Load to include the file in the DFM, Save to extract a previously saved file, and Clear to remove an embedded file. When there's a file loaded in this property, you don't need to ship the project file separately.

## **2.11 Data Aware Reports**

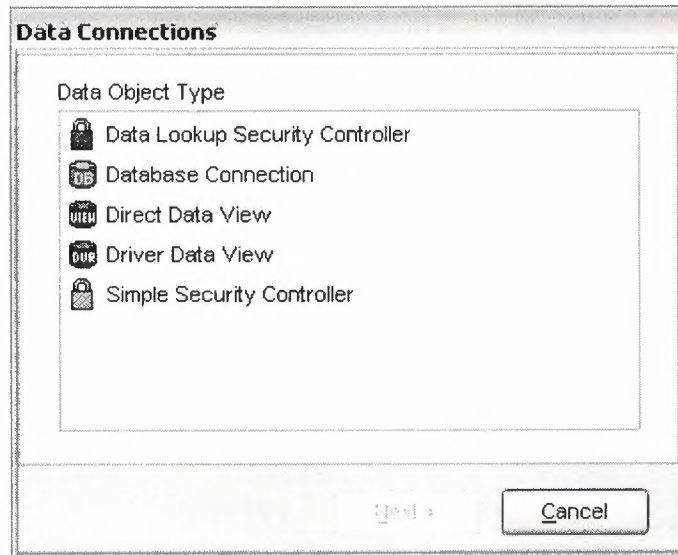
### **2.11.1 The Database Connection**

There are two ways to access data from inside a report: you can share the same connection established by your application, fetching records from Datasets that exists in your Forms or Datamodules, or you can configure a new connection on the report, allowing it to be independent of a particular application. For the first method you would use a Direct Data View and a Driver Data View for the second. Data View is the analog of a DataSource/DataSet combination inside the report.

If you intend to deploy your application using Nevrona's Rave Report Server, you should use Driver Data Views.

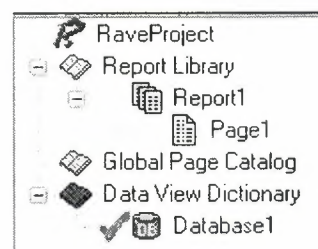
### **2.11.2 The Driver Data View**

Let's create a simple database report using a Driver Data View. Start the Rave Visual Designer, and start a new project. We need to define the database connection. To do this, choose File/New Database Object, or press the sixth button in the toolbar (the purple cube). The Data Connections window will appear:



**Figure 2.33** Data Connection Window

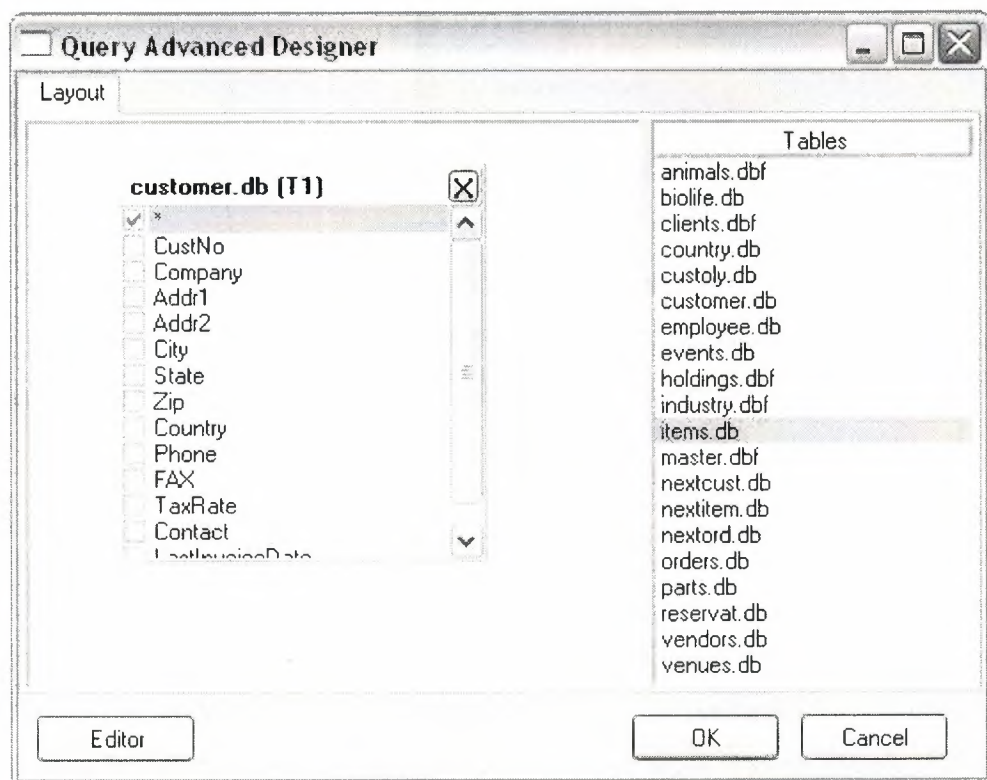
Choose Database Connection, and you will be asked which Data Link you are going to be using. There is a folder called DataLinks where Rave has been installed, containing some files with the .rwd extensions, responsible from the connection mechanism. By default, you can choose between BDE, DbExpress and ADO. I'll be using BDE for this example. Choose BDE; press Finish, and the Database Connection Parameters window will show up. Every Data Link has a different set of connection parameters available, similar to those available in the Delphi IDE. For now, just set Alias to DbDemos and press OK. Notice that a Database object has been added to the Project Treeview, under Data View Dictionary:



**Figure 2.34** Project Tree View

Notice that the settings you configured in the Database Connection Parameters, after the wizard, including username and password, if applicable, were saved in the AuthDesign property of the Database component. In the AuthRun property you can use different settings to be used at runtime, when your report has been deployed.

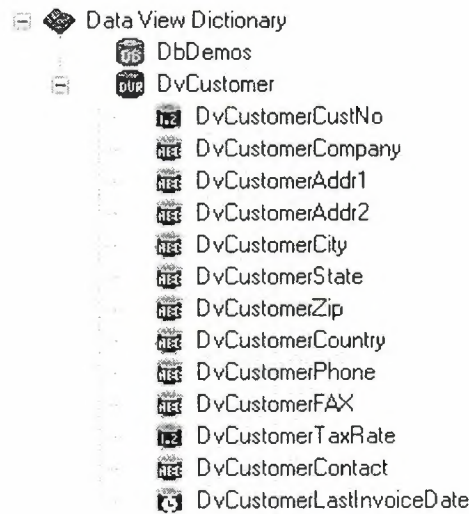
We are going to create now the Driver Data View. Click on New Data Object, and then choose Driver Data View. You should now choose the Database Connection that is going to be used by this Data View: choose the Database created in the previous step. A Query Advanced Designer will show up. Drag and Drop the table customer.db from the table list to the Layout window. It should look like this:



**Figure 2.35** Query Advanced Designer Window

If you have more than one table, you should drag and drop fields that should be joined between tables. If you press the Editor Button you can check the generated SQL, or type-in a more complex query. Let's keep the simple Customer Listing for now. Press OK and a DriverDataView will be added to the Project Treeview, below the Database components, having the selected fields as subitems:





**Figure 2.36** Project Tree View

Notice that I renamed the Database Connection and the Data View to more appropriate names. It's in the Treeview where properties of the fields should be set, like the Display Label (FullName property), and the DisplayFormat.

### 2.11.3 Regions and Bands

Report components that should be printed in a fixed position in every page, like fixed headers and footers can be put directly in page. Components, whose position will be dependent of previously printed items, should be put in bands. DataBands will be printed once for every record in the linked DataView, while regular Bands will only be printed once, regardless of how many records have been selected. Both can contain Data-Aware components (like DataText), or regular components (like Text).

Bands should be put inside Regions. Regions delimitate the width of the bands, and the maximum height that bands can use before starting a new page. One page can have many Regions, and one Region can contain many Bands.

Add a Region to the Page covering its whole area. Inside the region add a Band, to be used as the report header, a DataBand, to print the customer information, and another Band, the report footer.

If you wish to change the ordering of existing bands in a report, use the Move Forward and Move Behind buttons in the Alignment Toolbar.

Rename the bands to more meaningful names (I used Header, CustomerData and Footer). Set the DataView property of CustomerData to DvCustomer, and set CustomerData as the ControllerBand of the Header and Footer bands. You should also run the Band Style Editor, from the Object Inspector, and set the Print Location of those two bands to Body Header and Body Footer, respectively. You can have an idea on how the report is going to be printed observing the Band Display as you change the settings. It shows iterating bands repeated three times, and other bands only once:

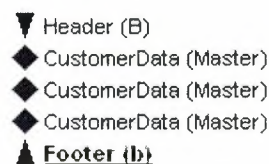


Figure 2.37 Band Display

We also want the Header to be printed in other pages in case the listing spans more than one page: check the New Page option in the Print Occurrence groupbox, in that same dialog.

The Footer band will only print when DvCustomers has reached its end. If you want it printed in every page, regardless of that, just put the components directly on the page, below the region, and not in a Band.

In the editor, you can quickly identify the relationship between bands, their styles and their print occurrences:

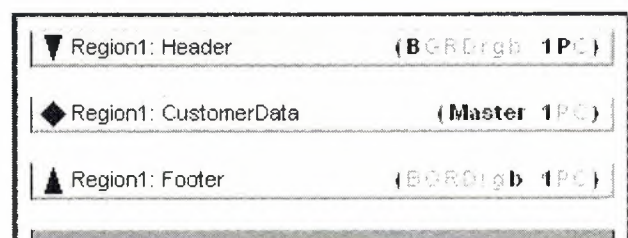


Figure 2.38 Editor Sample

#### **2.11.4 Adding Fields**

It's not hard to add fields to a report. You can Ctrl+Drag the fields from the DataView, in the Project Treeview, to add DataText components to the report, and Alt+Drag them to add Text components containing the Fullname property. This allows you to quickly create the layout of the report. Now add some fields to CustomerData and their title to the Header. I added CustNo, Company, Phone, TaxRate and LastInvoiceDate.

Don't forget that you can use the tools on the Alignment Toolbar to align the components, even if they are in different bands.

I added a title to the Header band and a simple text to the Footer band, indicating that the listing has ended. Later on the series we are going to see how to use the CalcOp and CalcTotal components to be able to add totals, averages and other calculated values to the Footer.

#### **2.11.5 Adding the Report to Your Project**

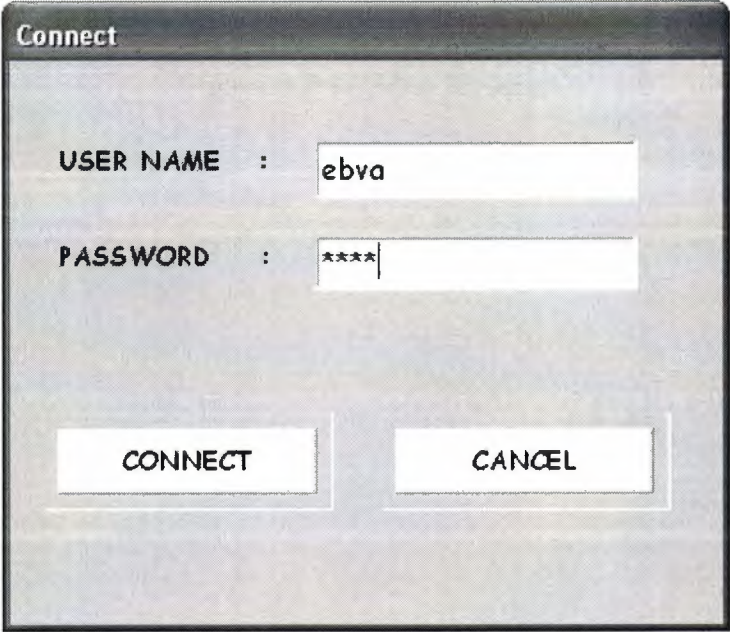
To add this report to your project you should use the same approach as seen in Part II: just use a RvProject in a Form or DataModule, link it to the report file, and call its Execute method. But there is one gotcha when using Driver Data Views: your application must load the appropriate driver. To do that, just add the unit RvDLBDE to your uses clause, if using BDE, RvDLDBX if using DbExpress, or RvDLADO if using ADO.

## CHAPTER THREE

### 3 STOCK PROPERTY BY USING DELPHI

#### 3.1 Database Connection Screen

When user executes program, first database connection screen appears. In this screen user enters user name and password to use the program. so user must have a valid user name and password. Also user must have appropriate privileges on database; such as view, add, update, delete.

A screenshot of a Delphi-style database connection dialog box titled "Connect". It features two input fields: "USER NAME" with the text "ebva" and "PASSWORD" with masked characters "\*\*\*\*". At the bottom are two buttons labeled "CONNECT" and "CANCEL".

Connect

USER NAME : ebva

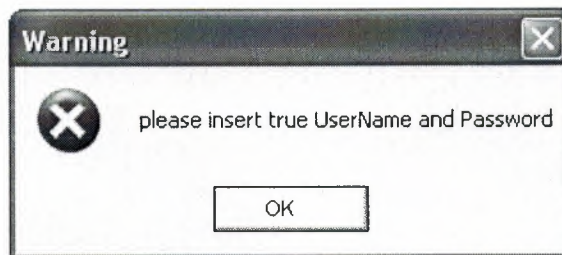
PASSWORD : \*\*\*\*

CONNECT CANCEL

Figure 3.1 Database Connection Screen



If user name or passwords are not entered correctly a screen appears with a message as "please insert true UserName and Password".



**Figure 3.2** Warning Message

### 3.2 Main Menu

When the user name and password are entered correctly user meets the Main Menu screen. As you can see in this figure there are 15 sections; house to let, house for sale, shop to let, shop for sale, plot for sale, garden for sale, building for sale, farm for sale, villa for sale, field for sale, flier print, about, informations, user register and exit are the names of the sections.

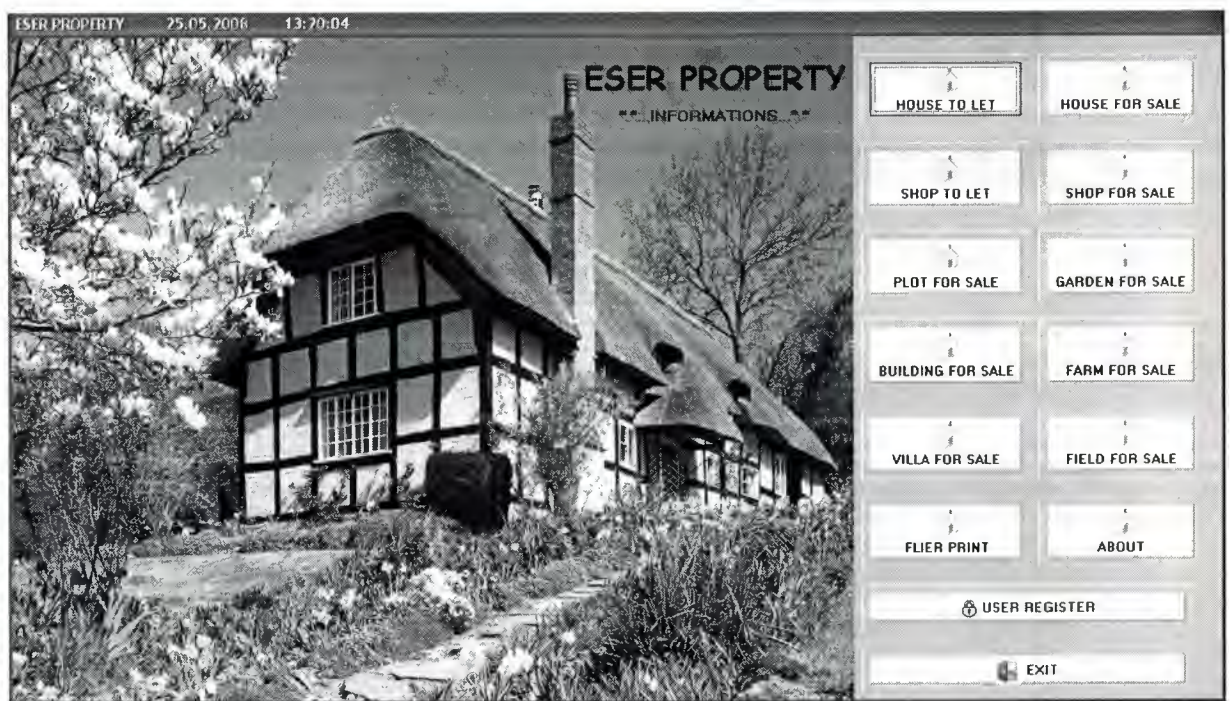


Figure 3.3 Main Menu



### 3.3 House to Let Menu

In house to let menu user can organize, search and print of house to let.

#### 3.3.1 House to let Organize Form

House to let organize form have 8 sections. The sections will be explain below.

New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already rented checkbox signed it means this house already rented otherwise if it is not signed it means it is available for let. Houseowner informations show the information of owner. Buyer informations show the information of customer. House to let informations show the information of house.

The screenshot shows a software window titled "house\_to\_let" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a toolbar at the top with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there are three main sections. The first section contains a "SEARCH" button with a magnifying glass icon, a checkbox labeled "Already rented" which is checked, and a "PRINT" button with a printer icon. The second section is titled "HOUSEOWNER INFORMATIONS" and contains three text input fields: "Name Surname : CEMALİYE DEMİR", "Cell Phone : 0533 765 34 32", and "Other Phone : 0392 223 12 45". The third section is titled "BUYER INFORMATIONS" and contains three text input fields: "Name Surname : AKAY ÇOLAK", "Cell Phone : 0 533 456 21 56", and "Other Phone : 0392 453 12 23". The fourth section is titled "HOUSE TO LET INFORMATIONS" and contains several text input fields: "Registration Date : 12/03/1996", "Price : 70.000,00 TL", "Square Meter : 125", "Aspect : KUZEY", "District : GÖNYELİ", "Floor : 3", "Type : STUDYO EV", "Heating System : SOBA", and "Address : ALİVEHİT SK.LEVENT APT. NO:12 LEFKOŞA/KKTC".

Figure 3.4 House to Let Organize Form

### 3.3.2 House to Let Search Form

House to let search form show to user detailed information and same time you can search the houses available for customer. Using preview button you can go to initial form.

Registrationdate	Squaremeter	District	Type	Price	Condition
12/03/1996	125	GÖNYELİ	STUDYO EV	70.000,00 TL	True
21/04/2006	260	ORTAKÖY	3+1	90.000,00 TL	False

Figure 3.5 House to Let Search Form



At house to let search part you can search available houses for letting according to their features. If the condition of house is false it means the house is empty you can let the house. If the condition is true it means you can not let the house because the house is already was letted.

housetolet\_search

RESEARCH

Search as to Sqare Meter : 260

Search as to District :

Search as to Price :

Search as to Heating System :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition
21/04/2006	260	ORTAKÖY	3+1	90.000,00 TL	False

**Figure 3.6** House to Let Search Form in Edit Mode

If you press previous section you can go to the current page of available house.



house\_to\_let

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ this house give to let PRINT

**HOUSEOWNER INFORMATION**

Name Surname : SEDA ÇİÇEK

Cell Phone : 0542 874 24 34

Other Phone : 0392 273 78 67

**HOUSE TO LET INFORMATION**

Registration Date : 21/04/2006 Price : 90.000,00 TL

Square Meter : 260 Aspect : KUZEY

District : ORTAKÖY Floor : 3

Type : 3+1 Heating System : DOĞALGAZ

Address : HUZURLU SK.SELEN APT.NO:7 LEFKOŞA/KKTC

Figure 3.7 House to Let Organize Form in Edit Mode

### 3.3.3 House to Let Report Form

Using house to let report form you can print the informations about that house.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, save, and other functions, along with a 'Close' button. The main content area displays the following information:

**ESER PROPERTY**

HOUSE TO LET

M <sup>2</sup>	:	260
TYPE	:	3+1
DISTRICT	:	ORTAKÖY
PRICE	:	90.000,00 TL
TELEPHONE	:	0000 000 00 00      9999 999 99 99

0% Page 1 of 1

**Figure 3.8** House to Let Report Form



### 3.4 House for Sale Menu

In house for sale menu user can organize, search and print of house for sale.

#### 3.4.1 House for Sale Organize Form

House for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this house already sold otherwise if it is not signed it means it is available for sale. Houseowner informations show the information of owner. Buyer informations show the information of customer. House for sale informations show the information of house.

The screenshot shows a software window titled "house\_for\_sale" with standard Windows window controls (minimize, maximize, close). The window contains several buttons at the top: NEW, PREVIOUS, NEXT, CLEAR, CANCEL, and SAVE. Below these buttons are two more buttons: SEARCH (with a magnifying glass icon) and PRINT (with a printer icon). A checkbox labeled "Already sold" is checked. The form is divided into three main sections: HOUSEOWNER INFORMATION, BUYER INFORMATION, and HOUSE FOR SALE INFORMATION. Each section contains several text input fields with labels and values.

HOUSEOWNER INFORMATION		BUYER INFORMATION	
Name Surname :	AHMET ESER	Name Surname :	EKREM PALTA
Cell Phone :	0532 234 56 98	Cell Phone :	0533 237 45 76
Other Phone :	0392 245 76 43	Other Phone :	0392 654 32 87

HOUSE FOR SALE INFORMATION			
Registration Date :	14/08/1990	Price :	85.000,00 TL
Square Meter :	225	Aspect :	BATI
District :	TAŞKINKÖY	Floor :	4
Type :	2+1	Heating System :	DOĞALGAZ
Address :	HANZADE SK.LEVENT APT.NO:12 LEFKOŞA/KKTC		

Figure 3.9 House for Sale Organize Form



### 3.4.2 House for Sale Search Form

House for sale search form show to user detailed information and same time you can search the houses available for customer.Using preview button you can go to initial form.

houseforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	14/08/1990	225	TAŞKINKÖY	2+1	85.000,00 TL	True	
	15/11/1999	145	HAMİTKÖY	3+1	66.000,00 TL	False	

Figure 3.10 House for Sale Search Form

At house for sale search part you can search available houses for selling according to their features. If the condition of house is false it means the house is empty you can sale the house. If the condition is true it means you can not sale the house because the house is already was sold.

houseforsale\_search


RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	
▶	15/11/1999	145	HAMİTKÖY	3+1	66.000,00 TL	False	^

**Figure 3.11** House for Sale Search in Edit Mode



If you press previous section you can go to the current page of available house.



**house\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ Sell house PRINT

**HOUSEOWNER INFORMATION**

Name Surname : MEHMET KAYA

Cell Phone : 0543 234 11 55

Other Phone : 0392 876 54 32

**HOUSE FOR SALE INFORMATION**

Registration Date : 15/11/1999 Price : 66.000,00 TL

Square Meter : 145 Aspect : DOĞU

District : HAMİTKÖY Floor : 2

Type : 3+1 Heating System : SOBA

Address : DEMİRLİ MH.ÇELSA SK.NO:23 LEFKOŞA/KKTC

**Figure 3.12** House for Sale Organize Form in Edit Mode

### 3.4.3 House for Sale Report Form

Using house to let report form you can print the informations about that house.

**Print Preview**

ESER PROPERTY

HOUSE FOR SALE

2  
M : 145

TYPE : 3+1

DISTRICT : HAMİTKÖY

PRICE : 66.000,00 TL

TELEPHONE : 0532 345 21 34 0542 843 77 59

0% Page 1 of 1

Figure 3.13 House for Sale Report



### 3.5 Shop to Let Menu

In shop to let menu user can organize, search and print of shop to let.

#### 3.5.1 Shop to Let Organize Form

Shop to let organize form have 8 sections. The sections will be explain below.

New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already rented checkbox signed it means this shop already rented otherwise if it is not signed it means it is available for let. Owner of a shop informations show the information of owner. Buyer informations show the information of customer. Shop to let informations show the information of shop.

The screenshot shows a software window titled 'shop\_to\_let'. At the top, there is a toolbar with buttons: NEW, PREVIOUS, NEXT, CLEAR, CANCEL, and SAVE. Below the toolbar, there is a 'SEARCH' button with a magnifying glass icon, a checkbox labeled 'Already rented' which is checked, and a 'PRINT' button with a printer icon. The main area is divided into three sections: 'OWNER OF A SHOP INFORMATIONS', 'BUYER INFORMATIONS', and 'SHOP TO LET INFORMATIONS'. Each section contains several text input fields for personal and property details.

OWNER OF A SHOP INFORMATIONS	
Name Surname :	TAHIR ÖZDEMİR
Cell Phone :	0542 865 45 67
Other Phone :	0392 865 34 54

BUYER INFORMATIONS	
Name Surname :	EVREN ÇAMLI
Cell Phone :	0533 764 89 21
Other Phone :	0392 751 39 30

SHOP TO LET INFORMATIONS			
Registration Date :	03/11/2003	Price :	45.000,00 TL
Square Meter :	55	Aspect :	BATI
District :	DEREBOYU	Floor :	1
Type :	PASAJ	Heating System :	DOĞALGAZ
Address :	SARMAŞIK SK.NO:5 LEFKOŞA/KKTC		

Figure 3.14 Shop to Let Organize Form

### 3.5.2 Shop to Let Search Form

Shop to let search form show to user detailed information and same time you can search the shops available for customer.Using preview button you can go to initial form.

**shoptolet\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	03/11/2003		55 DEREBOYU	PASAJ	45.000,00 TL	True	
□	30/01/1989		85 YENİKENT	GALERİ	45.000,00 TL	False	

Figure 3.15 Shop to Let Search Form



At shop to let search part you can search available shops for letting according to their features. If the condition of shop is false it means the shop is empty you can let the shop. If the condition is true it means you can not sale the shop because the shop is already was letted.

shoptolet\_search

RESEARCH

Search as to Square Meter : 85

Search as to District :

Search as to Price :

Search as to Heating System :

Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	
▶	30/01/1989	85	YENIKENT	GALERİ	45.000,00 TL	False	⌵

<
>

**Figure 3.16** Shop to Let Search Form in Edit Mode

If you press previous section you can go to the current page of available shop.



**shop\_to\_let**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ This shop give to let PRINT

**OWNER OF A SHOP INFORMATIONS**

Name Surname : CEM DÜNDAR

Cell Phone : 0533 245 67 87

Other Phone : 0392 754 34 90

**SHOP TO LET INFORMATIONS**

Registration Date : 30/01/1989 Price : 45.000,00 TL

Square Meter : 85 Aspect : GÜNEY

District : YENIKENT Floor : 1

Type : GALERİ Heating System : SOBA

Address : HELEGAN SK. NO:4 LEFKOŞA/KKTC

Figure 3.17 Shop to Let Organize Form in Edit Mode



### 3.5.3 Shop to Let Report Form

Using shop to let report form you can print the informations about that shops.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, copy, paste, and other functions. The main content area displays the following information:

**ESER PROPERTY**

**SHOP TO LET**

M <sup>2</sup>	:	85
TYPE	:	GALERİ
DISTRICT	:	YENİKENT
PRICE	:	45.000,00 TL
TELEPHONE	:	0532 345 21 34      0542 843 77 59

0% Page 1 of 1

Figure 3.18 Shop to Let Report

### 3.6.2 Shop for Sale Search Form

Shop for sale search form show to user detailed information and same time you can search the shops available for customer.Using preview button you can go to initial form.

Registrationdate	Squaremeter	District	Type	Price	Condition	
18/06/2002	55	GİRNEKAPI	ZEMİN	38.000,00 TL	True	
23/12/1996	75	KÜÇÜKKAYMAKL	ZEMİN	65.000,00 TL	False	

Figure 3.20 Shop for Sale Search Form

At shop for sale search part you can search available shops for selling according to their features.If the condition of shop is false it means the shop is empty you can sale the shop.If the condition is true it means you can not sale the shop because the shop is already was sold.

shopforsale\_search


RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 preview

Registrationdate	Squaremeter	District	Type	Price	Condition	
▶ 18/06/2002	55	GİRNEKAPI	ZEMİN	38.000,00 TL	True	^

Figure 3.21 Shop for Sale Search Form in Edit Mode



### 3.6 Shop for Sale Menu

In shop for sale menu user can organize, search and print of shop for sale.

#### 3.6.1 Shop for Sale Organize Form

Shop for sale organizes form have 8 sections. The sections will be explain below.  
New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this shop already sold otherwise if it is not signed it means it is available for sale. Owner of a shop informations show the information of owner. Buyer informations show the information of customer. Shop for sale informations show the information of shop.

The screenshot shows a Windows-style application window titled "shop\_for\_sale". The window contains several buttons at the top: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below these buttons are three main sections: a "SEARCH" button with a magnifying glass icon, a checkbox labeled "Already sold" which is checked, and a "PRINT" button with a printer icon. The form is divided into three main sections: "SHOPKEEPER INFORMATIONS", "BUYER INFORMATIONS", and "SHOP FOR SALE INFORMATIONS". Each section contains several text input fields for personal and business details.

SHOPKEEPER INFORMATIONS		BUYER INFORMATIONS	
Name Surname :	HÜSEYİN EREN	Name Surname :	YASİN KARLI
Cell Phone :	0543 285 23 18	Cell Phone :	0533 284 54 69
Other Phone :	0392 854 23 45	Other Phone :	0392 843 12 65

SHOP FOR SALE INFORMATIONS			
Registration Date :	18/06/2002	Price :	38.000,00 TL
Square Meter :	55	Aspect :	BATI
District :	GİRNEKAPI	Floor :	1
Type :	ZEMİN	Heating System :	DOĞALGAZ
Address :	YARALI SK.NO:9 LEFKOŞA/KKTC		

Figure 3.19 Shop for Sale Organize Form

If you press previous section you can go to the current page of available shop.

shop\_for\_sale

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☒ Already sold PRINT

**SHOPKEEPER INFORMATIONS**

Name Surname : HÜSEYİN EREN

Cell Phone : 0543 285 23 18

Other Phone : 0392 854 23 45

**BUYER INFORMATIONS**

Name Surname : YASİN KARLI

Cell Phone : 0533 284 54 69

Other Phone : 0392 843 12 65

**SHOP FOR SALE INFORMATIONS**

Registration Date : 18/06/2002 Price : 38.000,00 TL

Square Meter : 55 Aspect : BATI

District : GİRNEKAPI Floor : 1

Type : ZEMİN Heating System : DOĞALGAZ

Address : YARALI SK.NO:9 LEFKOŞA/KKTC

Figure 3.22 Shop for Sale Organize Form in Edit Mode



### 3.6.3 Shop for Sale Report Form

Using shop for sale report form you can print the informations about that shops.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, copy, paste, and other functions. The main content area displays a report form for 'ESER PROPERTY'. The form includes the following details:

<b>ESER PROPERTY</b>		
<b>SHOP FOR SALE</b>		
M <sup>2</sup>	:	75
TYPE	:	ZEMİN
DISTRICT	:	KÜÇÜKKAYMAKLI
PRICE	:	65.000,00 TL
TELEPHONE	:	0532 345 21 34      0542 843 77 59

At the bottom of the window, it says '0% Page 1 of 1'.

**Figure 3.23** Shop for Sale Report Form



### 3.7 Plot for Sale Menu

In plot for sale menu user can organize, search and print of plot for sale.

#### 3.7.1 Plot for Sale Organize Form

Plot for sale organizes form have 8 sections. The sections will be explain below.  
New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this plot already sold otherwise if it is not signed it means it is available for sale. Owner of a plot informations show the information of owner. Buyer informations show the information of customer. Plot for sale informations show the information of plot.

The screenshot shows a software window titled "plot\_for\_sale" with standard Windows window controls (minimize, maximize, close). The window contains a toolbar with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there is a "SEARCH" button with a magnifying glass icon and a "PRINT" button with a printer icon. A checkbox labeled "Already sold" is checked. The form is divided into three main sections: "OWNER OF A PLOT INFORMATIONS", "BUYER INFORMATIONS", and "PLOT FOR SALE INFORMATIONS".

OWNER OF A PLOT INFORMATIONS		BUYER INFORMATIONS	
Name Surname :	MURAT KARAHAN	Name Surname :	MAHIR SEREN
Cell Phone :	0542 645 39 65	Cell Phone :	0533 943 29 54
Other Phone :	0392 194 28 79	Other Phone :	0392 943 29 21

PLOT FOR SALE INFORMATIONS			
Registration Date :	27/10/1986	District :	SEFAKÖY
Square Meter :	320	Price :	85.000,00 TL
Type :	YOLÜSTÜ		
Address :	HASTANE KARŞISI HAŞMET SK. LEFKOŞA/KKTC		

Figure 3.24 Plot for Sale Organize Form

### 3.7.2 Plot for Sale Search Form

Plot for sale search form show to user detailed information and same time you can search the plots available for customer.Using preview button you can go to initial form.


**plotforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

 Preview

Registrationdate	Squaremeter	District	Type	Price	Condition	
▶ 27/10/1986	320	SEFAKÖY	YOLÜSTÜ	85.000,00 TL	True	^
15/02/2000	400	LEMAR	PARKYANI	95.000,00 TL	False	

Figure 3.25 Plot for Sale Search Form



At plot for sale search part you can search available plots for selling according to their features. If the condition of plot is false it means the plot is empty you can sale the plot. If the condition is true it means you can not sale the plot because the plot is already was sold.


**plotforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

 **Preview**

Registrationdate	Squaremeter	District	Type	Price	Condition	
▶ 15/02/2000	400	LEMAR	PARKYANI	95.000,00 TL	False	^

< >

**Figure 3.26** Plot for Sale Search Form in Edit Mode



If you press previous section you can go to the current page of available plot.

**plot\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ Sell plot PRINT

**OWNER OF A PLOT INFORMATIONS**

Name Surname : NURİ ERTAŞ

Cell Phone : 0533 296 33 65

Other Phone : 0392 743 21 84

**PLOT FOR SALE INFORMATIONS**

Registration Date : 15/02/2000 District : LEMAR

Square Meter : 400 Price : 95.000,00 TL


Type : PARKYANI

Address : FENERLİ SK. İLKÖĞRETİM KARŞISI LEFKOŞA/KKTC

**Figure 3.27** Plot for Sale Organize Form in Edit Mode

### 3.7.3 Plot for Sale Report Form

Using plot for sale report form you can print the informations about that plots.



The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for print, list, zoom, navigation, and a 'Close' button. The main content area displays the following information:

**ESER PROPERTY**

**PLOT FOR SALE**

M <sup>2</sup>	:	320
TYPE	:	YOLÜSTÜ
DISTRICT	:	SEFAKÖY
PRICE	:	85.000,00 TL
TELEPHONE	:	0532 345 21 34      0542 843 77 59

0% Page 1 of 1

**Figure 3.28** Plot for Sale Report Form



### 3.8 Garden for Sale Menu

In garden for sale menu user can organize, search and print of garden for sale.

#### 3.8.1 Garden for Sale Organize Form

Garden for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this garden already sold otherwise if it is not signed it means it is available for sale. Owner of a garden informations show the information of owner. Buyer informations show the information of customer. Garden for sale informations show the information of garden.

The screenshot shows a software window titled "garden\_for\_sale". At the top, there is a toolbar with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there are two buttons: "SEARCH" (with a magnifying glass icon) and "PRINT" (with a printer icon). In the center, there is a checkbox labeled "Already sold" which is checked. Below this, the form is divided into three main sections. The first section, "OWNER OF A GARDEN INFORMATIONS", contains three input fields: "Name Surname" with the value "SEFA ATACAN", "Cell Phone" with "0533 645 28 56", and "Other Phone" with "0392 754 38 59". The second section, "BUYER INFORMATIONS", also contains three input fields: "Name Surname" with "FATİH METE", "Cell Phone" with "0542 458 84 68", and "Other Phone" with "0392 734 29 59". The third section, "GARDEN FOR SALE INFORMATIONS", contains five input fields: "Registration Date" with "21/03/1990", "District" with "GÖNYELİ", "Square Meter" with "220", "Price" with "92.000,00 TL", and "Type" with "SERA". At the bottom, there is an "Address" field with the value "SEÇKİN SK. AKBANK KARŞISI LEFKOŞA/KKTC".

Figure 3.29 Garden for Sale Organize Form



### 3.8.2 Garden for Sale Search Form

Garden for sale search form show to user detailed information and same time you can search the gardens available for customer.Using preview button you can go to initial form.


**gardenforsale\_search**

**RESEARCH**

Search as to Sqare Meter :

Search as to District :

Search as to Price :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition ^
▶	21/03/1990	220	GÖNYELİ	SERA	92.000,00 TL	True
□	11/09/1992	350	KAYALI	410 KAYISI	88.000,00 TL	False

**Figure 3.30** Garden for Sale Search Form

At garden for sale search part you can search available gardens for selling according to their features. If the condition of garden is false it means the garden is empty you can sale the garden. If the condition is true it means you can not sale the garden because the garden is already was sold.

gardenforsale\_search

RESEARCH

Search as to Sqare Meter : 350

Search as to District :

Search as to Price :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition
11/09/1992	350	KAYALI	410 KAYISI	88.000,00 TL	False

**Figure 3.31** Garden for Sale Search Form in Edit Mode



If you press previous section you can go to the current page of available garden.



**garden\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH PRINT

☐ Sell garden

**OWNER OF A GARDEN INFORMATIONS**

Name Surname : DENIZ SAKAR

Cell Phone : 0542 743 94 98

Other Phone : 0392 732 18 39

**GARDEN FOR SALE INFORMATIONS**

Registration Date : 11/09/1992 District : KAYALI

Square Meter : 350 Price : 88.000,00 TL

Type : 410 KAYISI

Address : GÜZELYURT YOLU ÇAMLICA SK. LEFKOŞA/KKTC

**Figure 3.32** Garden for Sale Organize Form in Edit Mode



### 3.8.3 Garden for Sale Report Form

Using garden for sale report form you can print the informations about that gardens.

**ESER PROPERTY**

**GARDEN FOR SALE**

M <sup>2</sup>	:	350	
TYPE	:	410 KAYISI	
DISTRICT	:	KAYALI	
PRICE	:	88.000,00 TL	
TELEPHONE	:	0532 345 21 34	0542 843 77 59

0% Page 1 of 1

**Figure 3.33** Garden for Sale Report Form

### 3.9 Building For Sale Menu

In building for sale menu user can organize, search and print of building for sale.

#### 3.9.1 Building for Sale Organize Form

Building for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this building already sold otherwise if it is not signed it means it is available for sale. Owner of a building informations show the information of owner. Buyer informations show the information of customer. Building for sale informations show the information of building.

The screenshot shows a software window titled "building\_for\_sale" with standard Windows window controls (minimize, maximize, close). The window contains several sections for managing building sale applications.

**Navigation Buttons:** NEW, PREVIOUS, NEXT, CLEAR, CANCEL, SAVE.

**Search and Print Section:** A SEARCH button with a magnifying glass icon, a checkbox labeled "Already sold" which is checked, and a PRINT button with a printer icon.

**OWNER OF A BUILDING INFORMATIONS:**

- Name Surname : METİN ESENGÜL
- Cell Phone : 0542 456 67 85
- Other Phone : 0392 553 68 49

**BUYER INFORMATIONS:**

- Name Surname : MAHMUT YİĞİT
- Cell Phone : 0542 754 39 98
- Other Phone : 0392 943 18 45

**BUILDING FOR SALE INFORMATIONS:**

- Registration Date : 18/08/1995
- Square meter : 320
- District : METROPOL
- Type : 4+1
- Price : 75.000,00 TL
- Aspect : BATI
- Floor : 5
- Heating System : SOBA
- address : FAZİLET MH.SEREN SK.NO:7 LEFKOŞA/KKTC

Figure 3.34 Building for Sale Organize Form



### 3.9.2 Building for Sale Search Form

Building for sale search form show to user detailed information and same time you can search the buildings available for customer.Using preview button you can go to initial form.

**buildingforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

Search as to Heating System :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	18/08/1995	320	METROPOL	4+1	75.000,00 TL	True	
□	01/01/1993	185	BAHÇELİEVLI	3+1	98.765,00 TL	False	

Figure 3.35 Building for Sale Search Form



At building for sale search part you can search available buildings for selling according to their features. If the condition of building is false it means the building is empty you can sale the building. If the condition is true it means you can not sale the building because the building is already was sold.

buildingforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District :

Search as to Price :

75000

Search as to Heating System :

Preview

	Registrationdate	Squaremeter	District	Type	Price	Condition	
▶	18/08/1995	320	METROPOL	4+1	75.000,00 TL	True	^

**Figure 3.36** Building for Sale Search Form in Edit Mode

If you press previous section you can go to the current page of available building.

building\_for\_sale

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☒ Already sold PRINT

**OWNER OF A BUILDING INFORMATIONS**

Name Surname : METİN ESENGÜL

Cell Phone : 0542 456 67 85

Other Phone : 0392 553 68 49

**BUYER INFORMATIONS**

Name Surname : MAHMUT YİĞİT

Cell Phone : 0542 754 39 98

Other Phone : 0392 943 18 45

**BUILDING FOR SALE INFORMATIONS**

Registration Date : 18/08/1995 Price : 75.000,00 TL

Square meter : 320 Aspect : BATI

District : METROPOL Floor : 5

Type : 4+1 Heating System : SOBA

address : FAZİLET MH.SEREN SK.NO:7 LEFKOŞA/KKTC

Figure 3.37 Building for Sale Organize Form in Edit Mode

### 3.9.3 Building for Sale Report Form

Using building for sale report form you can print the informations about that buildings.

The screenshot shows a 'Print Preview' window with a toolbar at the top containing icons for back, forward, search, and other navigation functions, along with a 'Close' button. The main content area displays the following information:

**ESER PROPERTY**  
BUILDING FOR SALE

M <sup>2</sup>	:	320	
TYPE	:	4+1	
DISTRICT	:	METROPOL	
PRICE	:	75.000,00 TL	
TELEPHONE	:	0532 345 21 34	0542 843 77 59

0% Page 1 of 1

Figure 3.38 Building for Sale Report Form



### 3.10 Farm for Sale Menu

In farm for sale menu user can organize, search and print of farm for sale.

#### 3.10.1 Farm for Sale Organize Form

Farm for sale organize form have 8 sections. The sections will be explain below. New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this farm already sold otherwise if it is not signed it means it is available for sale. Owner of a farm informations show the information of owner. Buyer informations show the information of customer. Farm for sale informations show the information of farm.

The screenshot shows a software window titled "farm\_for\_sale". At the top, there is a toolbar with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there is a "SEARCH" button with a magnifying glass icon and a checkbox labeled "Already sold" which is checked. To the right of the checkbox is a "PRINT" button with a printer icon. The main area of the form is divided into three sections: "OWNER OF A FARM INFORMATIONS", "BUYER INFORMATIONS", and "FARM FOR SALE INFORMATIONS". Each section contains several input fields for text data.

OWNER OF A FARM INFORMATIONS		BUYER INFORMATIONS	
Name Surname :	SONER ERTEGÜL	Name Surname :	CIHAD SELVI
Cell Phone :	0533 574 29 59	Cell Phone :	0533 274 48 28
Other Phone :	0392 483 20 82	Other Phone :	0392941 26 54

FARM FOR SALE INFORMATIONS			
Registration Date :	13/12/1991	District :	HAMITKÖY
Square Meter :	3200	Price :	87.000,00 TL
Type :	BÜYÜKBAŞ		
Address :	CANDEMİR YOLU 8.KM'DE LEFKOŞA/KKTC		

Figure 3.39 Farm for Sale Organize Form

### 3.10.2 Farm for Sale Search Form

Farm for sale search form show to user detailed information and same time you can search the farms available for customer.Using preview button you can go to initial form.


**farmforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition	^
▶	13/12/1991	3200	HAMİTKÖY	BÜYÜKBAŞ	87.000,00 TL	True	
▶	29/03/1983	4000	DEMİRHAN	TAVUK	65.000,00 TL	False	

Figure 3.40 Farm for Sale Search Form







If you press previous section you can go to the current page of available farm.



**farm\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ sell farm PRINT

**OWNER OF A FARM INFORMATIONS**

Name Surname : AHMET SEYFI

Cell Phone : 0533 481 39 74

Other Phone : 0392 932 18 47

**FARM FOR SALE INFORMATIONS**

Registration Date : 29/03/1983 District : DEMIRHAN

Square Meter : 4000 Price : 65.000,00 TL

Type : TAVUK

Address : SANAYI YOLU 3.KM'DE LEFKOŞA/KKTC

**Figure 3.42** Farm for Sale Organize Form in Edit Mode

### 3.10.3 Farm for Sale Report Form

Using farm for sale report form you can print the informations about that farms.

**Print Preview**

Save Report

**ESER PROPERTY**

**FARM FOR SALE**

2  
M : 4000

TYPE : TAVUK

DISTRICT : DEMIRHAN

PRICE : 65.000,00 TL

TELEPHONE : 0532 345 21 34 0542 843 77 59

0% Page 1 of 1

**Figure 3.43** Farm for Sale Report Form



### 3.11 Villa for Sale Menu

In villa for sale menu user can organize, search and print of villa for sale.

#### 3.11.1 Villa for Sale Organize Form

Villa for sale organize form have 8 sections. The sections will be explain below.  
New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this villa already sold otherwise if it is not signed it means it is available for sale. Owner of a villa informations show the information of owner. Buyer informations show the information of customer. Villa for sale informations show the information of villa.

The screenshot shows a software window titled "villa\_for\_sale". At the top, there is a toolbar with buttons: NEW, PREVIOUS, NEXT, CLEAR, CANCEL, and SAVE. Below the toolbar, there is a SEARCH button with a magnifying glass icon, a checkbox labeled "Already sold" which is checked, and a PRINT button with a printer icon. The form is divided into three main sections: "OWNER OF A VILLA INFORMATIONS", "BUYER INFORMATIONS", and "VILLA FOR SALE INFORMATIONS".

OWNER OF A VILLA INFORMATIONS		BUYER INFORMATIONS	
Name Surname :	SEDAT YAREN	Name Surname :	ALI YILDIZ
Cell Phone :	0542 374 58 32	Cell Phone :	0533 852 37 19
Other Phone :	0392 963 87 21	Other Phone :	0392 563 95 64

VILLA FOR SALE INFORMATIONS			
Registration Date :	01/09/1998	District :	ORTAKÖY
Square Meter :	320	Price :	92.000,00 TL
Type :	DUBLEX		
Address :	SEFALİ MH. SİHIRLİ SK.NO:5 LEFKOŞA/KKTC		

Figure 3.44 Villa for Sale Organize Form




### 3.11.2 Villa for Sale Search Form

Villa for sale search form show to user detailed information and same time you can search the villas available for customer. Using preview button you can go to initial form.

**villaforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :   **Preview**

Search as to Price :

Registrationdate	Squaremeter	District	Type	Price	Condition	
▶ 01/09/1998	320	ORTAKÖY	DUBLEX	92.000,00 TL	True	^
17/04/2001	225	ERYAMAN	DUBLEX	99.000,00 TL	False	

**Figure 3.45** Villa for Sale Search Form

At villa for sale search part you can search available villas for selling according to their features. If the condition of villa is false it means the villa is empty you can sale the villa.If the condition is true it means you can not sale the villa because the villa is already was sold.

villaforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District : ORTAKÖY

Search as to Price :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition
01/09/1998	320	ORTAKÖY	DUBLEX	92.000,00 TL	True

**Figure 3.46** Villa for Sale Search Form in Edit Mode



If you press previous section you can go to the current page of available villa.

**villa\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☒ Already sold PRINT

**OWNER OF A VILLA INFORMATIONS**

Name Surname : SEDAT YAREN

Cell Phone : 0542 374 58 32

Other Phone : 0392 963 87 21

**BUYER INFORMATIONS**

Name Surname : ALI YILDIZ

Cell Phone : 0533 852 37 19

Other Phone : 0392 563 95 64

**VILLA FOR SALE INFORMATIONS**

Registration Date : 01/09/1998 District : ORTAKÖY

Square Meter : 320 Price : 92.000,00 TL

Type : DUBLEX

Address : SEFALI MH. SİHIRLİ SK.NO:5 LEFKOŞA/KKTC

Figure 3.47 Villa for Sale Organize Form in Edit Mode



### 3.11.3 Villa for Sale Report Form

Using villa for sale report form you can print the informations about that villas.

**Print Preview**

Print Preview window showing the Villa for Sale Report Form.

**ESER PROPERTY**

**VILLA FOR SALE**

M <sup>2</sup>	:	320	
TYPE	:	DUBLEX	
DISTRICT	:	ORTAKÖY	
PRICE	:	92.000,00 TL	
TELEPHONE	:	0532 345 21 34	0542 843 77 59

0% Page 1 of 1

## 3.12 Field for Sale Menu

In field for sale menu user can organize, search and print of field for sale.

### 3.12.1 Field for Sale Organize Form

Field for sale organize form have 8 sections. The sections will be explain below.  
New; create new application. Previous; you can call the previous application using this button. Next; you can call the next application using this button. Clear; with this button you can cancel the application. Cancel; with this button you can clear the application. Save; with this button you can save the application. Search; with this button you can search the application. Print; with this button you can print the application.

If already sold checkbox signed it means this field already sold otherwise if it is not signed it means it is available for sale. Owner of a field informations show the information of owner. Buyer informations show the information of customer. Field for sale informations show the information of field.

The screenshot shows a software window titled "field\_for\_sale". At the top, there is a toolbar with six buttons: "NEW", "PREVIOUS", "NEXT", "CLEAR", "CANCEL", and "SAVE". Below the toolbar, there is a "SEARCH" button with a magnifying glass icon and a checkbox labeled "Already sold" which is checked. To the right of the checkbox is a "PRINT" button with a printer icon. The main area of the form is divided into three sections. The first section, "OWNER OF A FIELD INFORMATIONS", contains three input fields: "Name Surname" with the value "CANER TOPBAŞ", "Cell Phone" with the value "0392 443 51 94", and "Other Phone" with the value "0392 443 51 94". The second section, "BUYER INFORMATIONS", also contains three input fields: "Name Surname" with the value "EMIRHAN CANPOLAT", "Cell Phone" with the value "0533 842 21 56", and "Other Phone" with the value "0392 932 17 43". The third section, "FIELD FOR SALE INFORMATIONS", contains five input fields: "Registration Date" with the value "19/03/1991", "District" with the value "KANLIDERE", "Square Meter" with the value "3400", "Price" with the value "86.000,00 TL", and "Type" with the value "217 ELMA". At the bottom, there is an "Address" field with the value "SANDIKLI YOLU 3.KM'DE LEFKOŞA/KKTC".

Figure 3.48 Field for Sale Organize Form



### 3.12.2 Field for Sale Search Form

Field for sale search form show to user detailed information and same time you can search the fields available for customer.Using preview button you can go to initial form.


**fieldforsale\_search**

**RESEARCH**

Search as to Square Meter :

Search as to District :

Search as to Price :

 **Preview**

	Registrationdate	Squaremeter	District	Type	Price	Condition	
<input checked="" type="checkbox"/>	19/03/1991	3400	KANLIDERE	217 ELMA	86.000,00 TL	True	
<input type="checkbox"/>	21/06/1990	4600	TAŞLIKÖY	KUYUSUZ	75.000,00 TL	False	
<input type="checkbox"/>	21/05/1989	4600	TAŞLIKÖY	105 KAYISI	78.000,00 TL	False	
<input type="checkbox"/>	05/11/2004	2500	DEĞİRMELİK	KUYULU	75.000,00 TL	False	

**Figure 3.49** Field for Sale Search Form



At field for sale search part you can search available fields for selling according to their features. If the condition of field is false it means the field is empty you can sale the field. If the condition is true it means you can not sale the field because the field is already was sold.

fieldforsale\_search

RESEARCH

Search as to Square Meter :

Search as to District : DEĞİRMENLİK

Search as to Price :

Preview

Registrationdate	Squaremeter	District	Type	Price	Condition	
05/11/2004	2500	DEĞİRMENLİK	KUYULU	75.000,00 TL	False	

**Figure 3.50** Field for Sale Search Form in Edit Mode

If you press previous section you can go to the current page of available field.



**field\_for\_sale**

NEW PREVIOUS NEXT CLEAR CANCEL SAVE

SEARCH ☐ Sell field PRINT

**OWNER OF A FIELD INFORMATIONS**

Name Surname : DURMUŞ TEZCAN

Cell Phone : 0392 483 45 68

Other Phone : 0392 483 45 68

**FIELD FOR SALE INFORMATIONS**

Registration Date : 05/11/2004 District : DEĞİRMENLİK

Square Meter : 2500 Price : 75.000,00 TL

Type : KUYULU

Address : HAVAALANI YOLU 8.KM'DE LEFKOŞA/KKTC

Figure 3.51 Field for Sale Organize Form in Edit Mode

### 3.12.3 Field for Sale Report Form

Using field for sale report form you can print the informations about that fields.

**Print Preview**

Field for Sale Report Form

**ESER PROPERTY**

**FIELD FOR SALE**

2  
M : 2500

TYPE : KUYULU

DISTRICT : DEĞİRMENLİK

PRICE : 75.000,00 TL

TELEPHONE : 0532 345 21 34 0542 843 77 59

0% Page 1 of 1

**Figure 3.52** Field for Sale Report Form



### 3.13 Flier Print Menu

In flier print menu user can print all advertisements.

#### 3.13.1 Flier Print Organize Form

In flier print organize form user can print all advertisements related to each type houses, shops, villas, plots, fields, gardens, buildings and farms.

Using preview button you can go to main menu.

flier\_print

# ESER PROPERTY

 HOUSE TO LET ADVERTISEMENTS	 VILLA FOR SALE ADVERTISEMENTS
 SHOP TO LET ADVERTISEMENTS	 PLOT FOR SALE ADVERTISEMENTS
 HOUSE FOR SALE ADVERTISEMENTS	 FIELD FOR SALE ADVERTISEMENTS
 SHOP FOR SALE ADVERTISEMENTS	 GARDEN FOR SALE ADVERTISEMENTS
 BUILDING FOR SALE ADVERTISEMENTS	 FARM FOR SALE ADVERTISEMENTS

 PREVIEW

Figure 3.53 Flier Print Form

### 3.13.2 House to Let Advertisements Form

Using this form you can print the advertisements about that house to let.

**ESER PROPERTY**

**HOUSES TO LET**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
125	GÖNYELİ	STUDYO EV	70.000,00 TL
260	ORTAKÖY	3+1	90.000,00 TL

Page 1 of 1

**Figure 3.54** House to Let Advertisements Form

### 3.13.3 Villa for Sale Advertisements Form

Using this form you can print the advertisements about that villa for sale.



**ESER PROPERTY  
VILLAS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
320	ORTAKÖY	DUBLEX	92.000,00 TL
225	ERYAMAN	DUBLEX	98.000,00 TL

Page 1 of 1

**Figure 3.55** Villa for Sale Advertisements Form



3.13.4 Shop to Let Advertisements Form

Using this form you can print the advertisements about that shop to let.

Print Preview

Close

# ESER PROPERTY SHOPS TO LET

TELEPHONE : 0000 000 00 00 9999 999 99 99

SQUARE METER	DISTRICT	TYPE	PRICE
55	DEREBOYU	PASAJ	45.000,00 TL
85	YENIKENT	GALERI	45.000,00 TL

Page 1 of 1

Figure 3.56 Shop to Let Advertisements Form

### 3.13.5 Plot for Sale Advertisements Form

Using this form you can print the advertisements about that plot for sale.



**ESER PROPERTY**

**PLOTS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
320	SEFAKÖY	YOLÜSTÜ	85.000,00 TL
400	LEMAR	PARKYANI	95.000,00 TL

Page 1 of 1

**Figure 3.57** Plot for Sale Advertisements Form

### 3.13.6 House for Sale Advertisements Form

Using this form you can print the advertisements about that house for sale.

**ESER PROPERTY**

**HOUSES FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
225	TAŞKINKÖY	2+1	85.000,00 TL
145	HAMİTKÖY	3+1	66.000,00 TL

Page 1 of 1

**Figure 3.58** House for Sale Advertisements Form



### 3.13.7 Field for Sale Advertisements Form

Using this form you can print the advertisements about that field for sale.

The image shows a 'Print Preview' window for a form titled 'ESER PROPERTY FIELDS FOR SALE'. The form includes a telephone number and a table of properties for sale.

**ESER PROPERTY  
FIELDS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

S Q U A R E M E T E R	D I S T R I C T	T Y P E	P R I C E
3400	KANLIDERE	217 ELMA	88.000,00 TL
4600	TAŞLIKÖY	KUYUSUZ	75.000,00 TL
4600	TAŞLIKÖY	105 KAYISI	78.000,00 TL
2500	DEĞİRMELİK	KUYULU	75.000,00 TL

Page 1 of 1

**Figure 3.59** Field for Sale Advertisements Form

### 3.13.8 Shop for Sale Advertisements Form

Using this form you can print the advertisements about that shop for sale.

**ESER PROPERTY  
SHOPS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
55	GİRNEKAPI	ZEMİN	38.000,00 TL
75	KÜÇÜKKAYMAKLI	ZEMİN	65.000,00 TL

Page 1 of 1

**Figure 3.60** Shop for Sale Advertisements Form

**3.13.9 Garden for Sale Advertisements Form**

Using this form you can print the advertisements about that garden for sale.

**Print Preview**

Close

# ESER PROPERTY GARDENS FOR SALE

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	PRICE
220	GÖNYELİ	92.000,00 TL
360	KAYALI	88.000,00 TL

Page 1 of 1

**Figure 3.61** Garden for Sale Advertisements Form



### 3.13.10 Building for Sale Advertisements Form

Using this form you can print the advertisements about that building for sale.

**ESER PROPERTY**

**BUILDINGS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

SQUARE METER	DISTRICT	TYPE	PRICE
320	METROPOL	4+1	75.000,00 TL
185	BAHÇELİEVLER	3+1	98.785,00 TL

Page 1 of 1

**Figure 3.62** Building for Sale Advertisements Form

### 3.13.11 Farm for Sale Advertisements Form

Using this form you can print the advertisements about that farm for sale.

**ESER PROPERTY  
FARMS FOR SALE**

**TELEPHONE : 0000 000 00 00 9999 999 99 99**

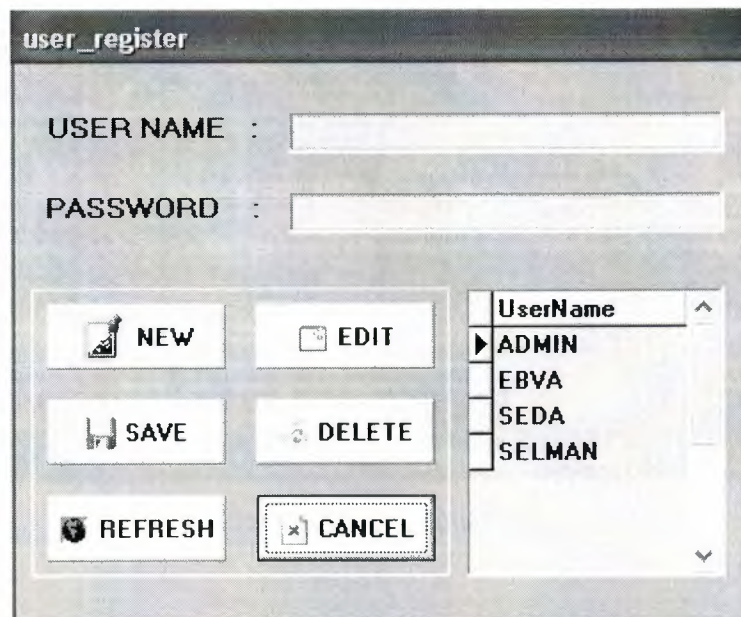
SQUARE METER	DISTRICT	TYPE	PRICE
3200	HAMITKÖY	BÜYÜKBAŞ	87.000,00 TL
4000	DEMİRHAN	TAVUK	65.000,00 TL

Page 1 of 1

**Figure 3.63** Farm for Sale Advertisements Form

### 3.14 User Register Menu

When you press the user register button you are going to open new form here who will use this program can be registered and they can use the program and same time you can exchange your password.If you press new button new admin can be added to user list.If you press edit button you can change your informations.If you press save button you can save your informations.If you press delete button you can deleted.User information from system.If you press refresh button you can clean the page.If you press cancel button you can leave this form and you can go to main menu.



The screenshot shows a window titled "user\_register". It has two text input fields: "USER NAME" and "PASSWORD". Below the inputs are six buttons arranged in a 3x2 grid: "NEW" (with a plus icon), "EDIT" (with a document icon), "SAVE" (with a floppy disk icon), "DELETE" (with a trash can icon), "REFRESH" (with a circular arrow icon), and "CANCEL" (with an 'X' icon). To the right of the buttons is a list box titled "UserName" with a scroll bar. The list contains four items: "ADMIN", "EBVA", "SED", and "SELMAN".

Figure 3.64 User Register From



### 3.15 About Menu

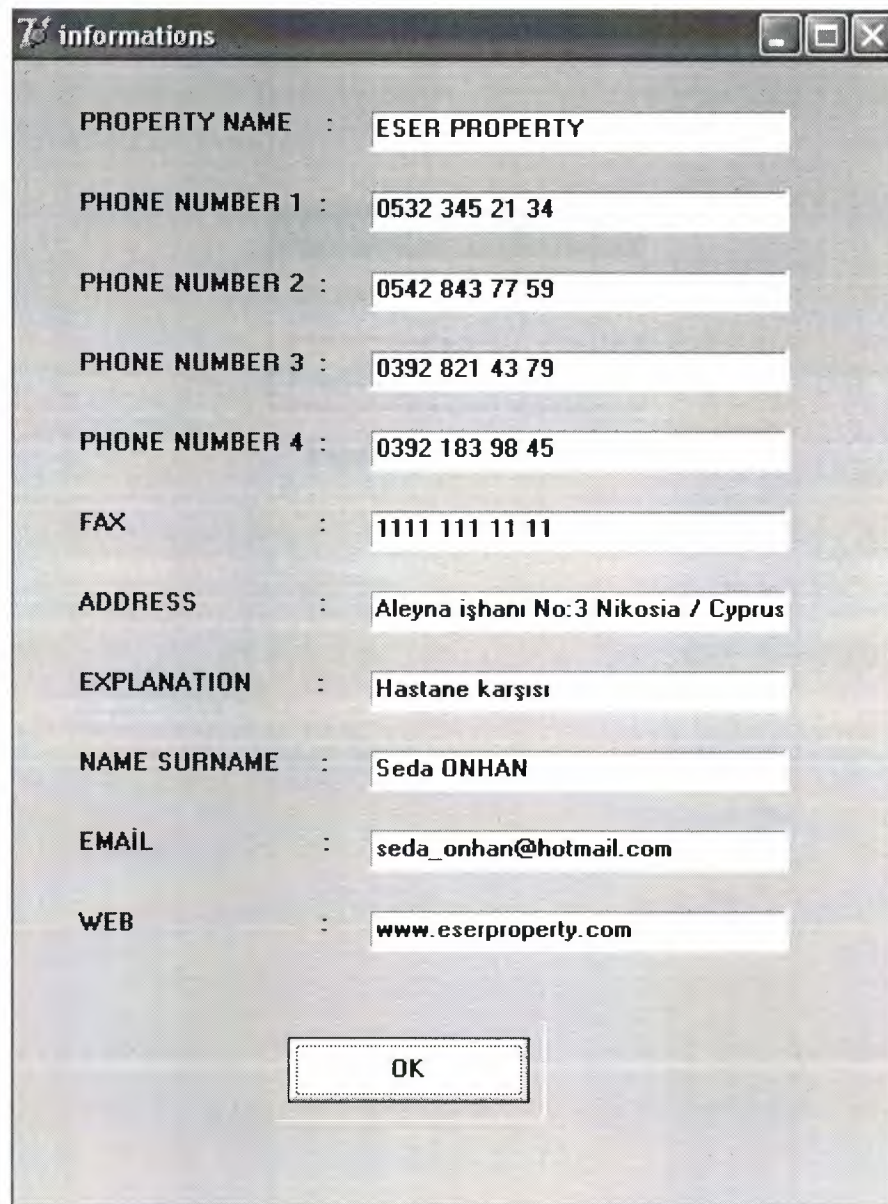
This form gives informations about the current program and owner of this program.



**Figure 3.65** About Form

### 3.16 Informations Menu

Using informations menu you can get all informations about the property.



The image shows a software window titled "informations" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a form with the following fields and values:

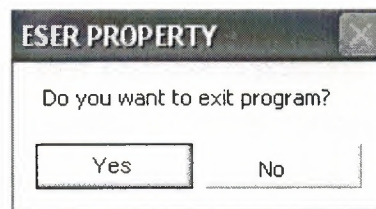
Field	Value
PROPERTY NAME	ESER PROPERTY
PHONE NUMBER 1	0532 345 21 34
PHONE NUMBER 2	0542 843 77 59
PHONE NUMBER 3	0392 821 43 79
PHONE NUMBER 4	0392 183 98 45
FAX	1111 111 11 11
ADDRESS	Aleyna işhanı No:3 Nikosia / Cyprus
EXPLANATION	Hastane karşısı
NAME SURNAME	Seda ONHAN
EMAIL	seda_onhan@hotmail.com
WEB	www.eserproperty.com

At the bottom center of the window is an "OK" button.

Figure 3.66 Informations Form

### 3.17 Exit Menu

When you click the exit menu ( yes / no ) you can decide to continue search or exit the program.



**Figure 3.67** Exit Form



## CONCLUSION

In this Graduation Project stock program for any property using Delphi was examined.

This program can be used easily for each user that can record customer information.

The operation structures of this program could be explained briefly; as follows when user executes program, first database connection screen appears. In this screen user enters user name and password to use the program. so user must have a valid user name and password. Also user must have appropriate privileges on database; such as view, add, update, delete.

When the user name and password are entered correctly user meets the Main Menu screen. As you can see in this figure there are 15 sections; house to let, house for sale, shop to let, shop for sale, plot for sale, garden for sale, building for sale, farm for sale, villa for sale, field for sale, flier print, about, informations, user register and exit are the names of the sections.

For future implementations the current program can be developed using different program languages.

## REFERENCES

<http://www.codegear.com>

[http://www.scalabium.com/faq/dc\\_tips.htm](http://www.scalabium.com/faq/dc_tips.htm)

<http://www.nevrona.com/>

Delphi Programming Explorer, Jeff Dontemann – Jim Mischel ISBN 1-883-57725-X

Database Application Developers Book for Delphi (e Book)

Borland Delphi 6 for Windows (e Book)

Mastering Delphi 6 – Marco Cantu

## APPENDIX

### Program Codes

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, jpeg, ExtCtrls, ImgList, ComCtrls, ToolWin;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Label1: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn7: TBitBtn;
    BitBtn8: TBitBtn;
    BitBtn9: TBitBtn;
    BitBtn10: TBitBtn;
    BitBtn11: TBitBtn;
    BitBtn12: TBitBtn;
    BitBtn13: TBitBtn;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    Bevel5: TBevel;
    Bevel6: TBevel;
    Bevel7: TBevel;
    Bevel8: TBevel;
    Bevel9: TBevel;
    Bevel10: TBevel;
    Bevel11: TBevel;
    Bevel12: TBevel;
    Timer1: TTimer;
    ImageList1: TImageList;
    ImageList2: TImageList;
    Label2: TLabel;
    BitBtn14: TBitBtn;
    Bevel13: TBevel;
    Bevel14: TBevel;
    procedure BitBtn1Click(Sender: TObject);
```



```

procedure BitBtn2Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure BitBtn5Click(Sender: TObject);
procedure BitBtn6Click(Sender: TObject);
procedure BitBtn9Click(Sender: TObject);
procedure BitBtn7Click(Sender: TObject);
procedure BitBtn8Click(Sender: TObject);
procedure BitBtn10Click(Sender: TObject);
procedure BitBtn11Click(Sender: TObject);
procedure BitBtn12Click(Sender: TObject);
procedure BitBtn13Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure Label2DbClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure BitBtn14Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

  uses unit2, unit5, unit8, unit10, unit6, unit3, unit4, unit9, unit7,
    unit11, unit22, unit23, unit44, unit45, unit47;

{$R *.dfm}

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  housetolet.ShowModal;
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
  shoptolet.ShowModal;
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
begin
  plot.ShowModal;
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
begin
  building.ShowModal;
end;

```

```

end;

procedure TForm1.BitBtn5Click(Sender: TObject);
begin
villa.ShowModal;
end;

procedure TForm1.BitBtn6Click(Sender: TObject);
begin
flierprint.ShowModal;
end;

procedure TForm1.BitBtn9Click(Sender: TObject);
begin
garden.ShowModal;
end;

procedure TForm1.BitBtn7Click(Sender: TObject);
begin
houseforsale.ShowModal;
end;

procedure TForm1.BitBtn8Click(Sender: TObject);
begin
shopforsale.ShowModal;
end;

procedure TForm1.BitBtn10Click(Sender: TObject);
begin
farm.ShowModal;
end;

procedure TForm1.BitBtn11Click(Sender: TObject);
begin
field.ShowModal;
end;

procedure TForm1.BitBtn12Click(Sender: TObject);
begin
about.ShowModal;
end;

procedure TForm1.BitBtn13Click(Sender: TObject);
begin
if(Application.MessageBox('Do you want to exit program?', 'ESER
PROPERTY', MB_YESNO)=IDYES)then
halt;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin

```

```
form1.caption:='ESER PROPERTY      '+DateTOStr(now)+'      '+TIMETostr(now)+'      ';
end;
```

```
procedure TForm1.Label2DbClick(Sender: TObject);
begin
informations.Show;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
Form1.ClientHeight:=599;
Form1.ClientWidth:=1072;
end;
```

```
procedure TForm1.BitBtn14Click(Sender: TObject);
begin
form47.ShowModal;
end;
```

```
end.
```

```
unit Unit2;
```

```
interface
```

```
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, DBCtrls, ExtCtrls, Buttons, ImgList, ComCtrls, ToolWin,
Mask;
```

```
type
Thousetolet = class(TForm)
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
ToolBar1: TToolBar;
ToolButton1: TToolButton;
ToolButton2: TToolButton;
ToolButton3: TToolButton;
ToolButton4: TToolButton;
ToolButton5: TToolButton;
ToolButton6: TToolButton;
ToolButton7: TToolButton;
ToolButton8: TToolButton;
ToolButton9: TToolButton;
ToolButton10: TToolButton;
ToolButton11: TToolButton;
ImageList1: TImageList;
```



```

BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
Bevel1: TBevel;
Bevel2: TBevel;
DBCheckBox1: TDBCheckBox;
Bevel3: TBevel;
Bevel4: TBevel;
Bevel5: TBevel;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure ToolButton9Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure FormCreate(Sender: TObject);

```

```

private
  { Private declarations }
public
  { Public declarations }
end;

var
  housetolet: Thousetolet;

implementation

  uses unit45, unit12, unit24;

  {$R *.dfm}

  procedure Thousetolet.ToolButton1Click(Sender: TObject);
  begin
    dm.tkhouse.Insert;
  end;

  procedure Thousetolet.ToolButton3Click(Sender: TObject);
  begin
    dm.tkhouse.Prior;
  end;

  procedure Thousetolet.ToolButton5Click(Sender: TObject);
  begin
    dm.tkhouse.Next;
  end;

  procedure Thousetolet.ToolButton7Click(Sender: TObject);
  begin
    dm.tkhouse.Cancel;
  end;

  procedure Thousetolet.BitBtn1Click(Sender: TObject);
  begin
    housetoletsearch.ShowModal;
  end;

  procedure Thousetolet.BitBtn2Click(Sender: TObject);
  begin
    housetoletreport.QuickRep1.Preview;
  end;

  procedure Thousetolet.DBCheckBox1Click(Sender: TObject);
  begin
    if DBCheckBox1.Checked=true then
    begin
      GroupBox2.Visible:=true;
      DBCheckBox1.Caption:='Already rented';
    end;
  end;

```

```

end;
if DBCheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='this house give to let';
end;
end;

procedure Thousetolet.ToolButton11Click(Sender: TObject);
begin
dm.tkhouse.Edit;
dm.tkhouse.Post;
ShowMessage('Record is registered');
end;

procedure Thousetolet.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Thousetolet.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Thousetolet.ToolButton9Click(Sender: TObject);
begin
try
if(Application.MessageBox('Record will be delete are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tkhouse.Delete;
except
ShowMessage('Cant delete empty record!');
end;
end;

procedure Thousetolet.FormKeyPress(Sender: TObject; var Key: Char);
begin
If (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Thousetolet.FormCreate(Sender: TObject);
begin
housetolet.ClientHeight:=606;
housetolet.ClientWidth:=695;
end;

```



end.

unit Unit3;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, ImgList, ComCtrls, ToolWin, StdCtrls, Mask, DBCtrls, Buttons,  
ExtCtrls;

type

Tshopforsale = class(TForm)  
  ToolBar1: TToolBar;  
  ToolButton1: TToolButton;  
  ToolButton2: TToolButton;  
  ToolButton3: TToolButton;  
  ToolButton4: TToolButton;  
  ToolButton5: TToolButton;  
  ToolButton6: TToolButton;  
  ToolButton7: TToolButton;  
  ToolButton8: TToolButton;  
  ToolButton9: TToolButton;  
  ToolButton10: TToolButton;  
  ToolButton11: TToolButton;  
  ImageList1: TImageList;  
  Bevel1: TBevel;  
  Bevel2: TBevel;  
  Bevel3: TBevel;  
  BitBtn1: TBitBtn;  
  BitBtn2: TBitBtn;  
  Bevel4: TBevel;  
  Bevel5: TBevel;  
  DBCheckBox1: TDBCheckBox;  
  GroupBox1: TGroupBox;  
  GroupBox2: TGroupBox;  
  GroupBox3: TGroupBox;  
  Label1: TLabel;  
  Label2: TLabel;  
  Label3: TLabel;  
  Label4: TLabel;  
  Label5: TLabel;  
  Label6: TLabel;  
  Label7: TLabel;  
  Label8: TLabel;  
  Label9: TLabel;  
  Label10: TLabel;  
  Label11: TLabel;  
  Label12: TLabel;

```

Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
procedure DBCheckBox1Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  shopforsale: Tshopforsale;

implementation

  uses unit45, unit15, unit27;

{$R *.dfm}

procedure Tshopforsale.DBCheckBox1Click(Sender: TObject);
begin
  if DBCheckBox1.Checked=true then
  begin
    GroupBox2.Visible:=true;

```

```

DBCheckBox1.Caption:='Already sold';
end;
if DBCheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell shop';
end;
end;

procedure Tshopforsale.ToolButton11Click(Sender: TObject);
begin
dm.tsshop.Edit;
dm.tsshop.Post;
ShowMessage('Record is registered');
end;

procedure Tshopforsale.ToolButton1Click(Sender: TObject);
begin
dm.tsshop.Insert;
end;

procedure Tshopforsale.ToolButton3Click(Sender: TObject);
begin
dm.tsshop.Prior;
end;

procedure Tshopforsale.ToolButton5Click(Sender: TObject);
begin
dm.tsshop.Next;
end;

procedure Tshopforsale.ToolButton7Click(Sender: TObject);
begin
dm.tsshop.Cancel;
end;

procedure Tshopforsale.BitBtn1Click(Sender: TObject);
begin
shopforsalesearch.ShowModal;
end;

procedure Tshopforsale.BitBtn2Click(Sender: TObject);
begin
shopforsalereport.QuickRep1.Preview;
end;

procedure Tshopforsale.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

```



```

procedure Tshopforsale.DBEdit1Exit(Sender: TObject);
begin
  if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Tshopforsale.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key = #13) then
  begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
  end;

end;

procedure Tshopforsale.ToolButton9Click(Sender: TObject);
begin
  try
    if (Application.MessageBox('Record will be deleted are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tsshop.Delete;
  except
    ShowMessage('Cant delete empty record!');
  end;

end;

procedure Tshopforsale.FormCreate(Sender: TObject);
begin
shopforsale.ClientHeight:=608;
shopforsale.ClientWidth:=695;
end;

end.

unit Unit4;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,
  ToolWin;

type
  Thouseforsale = class(TForm)
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;

```

ToolButton2: TToolButton;  
 ToolButton3: TToolButton;  
 ToolButton4: TToolButton;  
 ToolButton5: TToolButton;  
 ToolButton6: TToolButton;  
 ToolButton7: TToolButton;  
 ToolButton8: TToolButton;  
 ToolButton9: TToolButton;  
 ToolButton10: TToolButton;  
 ToolButton11: TToolButton;  
 ImageList1: TImageList;  
 Bevel1: TBevel;  
 Bevel2: TBevel;  
 BitBtn1: TBitBtn;  
 BitBtn2: TBitBtn;  
 DBCheckBox1: TDBCheckBox;  
 Bevel3: TBevel;  
 Bevel4: TBevel;  
 Bevel5: TBevel;  
 GroupBox1: TGroupBox;  
 GroupBox2: TGroupBox;  
 GroupBox3: TGroupBox;  
 Label1: TLabel;  
 Label2: TLabel;  
 Label3: TLabel;  
 Label4: TLabel;  
 Label5: TLabel;  
 Label6: TLabel;  
 Label7: TLabel;  
 Label8: TLabel;  
 Label9: TLabel;  
 Label10: TLabel;  
 Label11: TLabel;  
 Label12: TLabel;  
 Label13: TLabel;  
 Label14: TLabel;  
 Label15: TLabel;  
 DBEdit1: TDBEdit;  
 DBEdit2: TDBEdit;  
 DBEdit3: TDBEdit;  
 DBEdit4: TDBEdit;  
 DBEdit5: TDBEdit;  
 DBEdit6: TDBEdit;  
 DBEdit7: TDBEdit;  
 DBEdit8: TDBEdit;  
 DBEdit9: TDBEdit;  
 DBEdit10: TDBEdit;  
 DBEdit11: TDBEdit;  
 DBEdit12: TDBEdit;  
 DBEdit13: TDBEdit;  
 DBEdit14: TDBEdit;

```

DBEdit15: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  houseforsale: Thouseforsale;

implementation

  uses unit45, unit14, unit26;

{$R *.dfm}

procedure Thouseforsale.ToolButton1Click(Sender: TObject);
begin
  dm.tshouse.Insert;
end;

procedure Thouseforsale.ToolButton3Click(Sender: TObject);
begin
  dm.tshouse.Prior;
end;

procedure Thouseforsale.ToolButton5Click(Sender: TObject);
begin
  dm.tshouse.Next;
end;

procedure Thouseforsale.ToolButton7Click(Sender: TObject);
begin
  dm.tshouse.Cancel;
end;

procedure Thouseforsale.ToolButton11Click(Sender: TObject);
begin

```



```

dm.tshouse.Edit;
dm.tshouse.Post;
ShowMessage('Record is registered');
end;

procedure Thouseforsale.BitBtn1Click(Sender: TObject);
begin
houseforsalesearch.ShowModal;
end;

procedure Thouseforsale.BitBtn2Click(Sender: TObject);
begin
houseforsalereport.QuickRep1.Preview;
end;

procedure Thouseforsale.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already sold';
end;
if DBcheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell house';
end;
end;

procedure Thouseforsale.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Thouseforsale.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Thouseforsale.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

procedure Thouseforsale.ToolButton9Click(Sender: TObject);
begin

```

```

try
if (Application.MessageBox('Record wii be deleted are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tshouse.Delete;
except
  ShowMessage('Cant delete empty record!');
end;

```

```

end;

```

```

procedure Thouseforsale.FormCreate(Sender: TObject);
begin
houseforsale.ClientHeight:=609;
houseforsale.ClientWidth:=695;
end;

```

```

end.

```

```

unit Unit5;

```

```

interface

```

```

uses

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,
  ToolWin;

```

```

type

```

```

Tshoptolet = class(TForm)
  ToolBar1: TToolBar;
  ToolButton1: TToolButton;
  ToolButton2: TToolButton;
  ToolButton3: TToolButton;
  ToolButton4: TToolButton;
  ToolButton5: TToolButton;
  ToolButton6: TToolButton;
  ToolButton7: TToolButton;
  ToolButton8: TToolButton;
  ToolButton9: TToolButton;
  ToolButton10: TToolButton;
  ToolButton11: TToolButton;
  ImageList1: TImageList;
  Bevel1: TBevel;
  Bevel2: TBevel;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  DBCheckBox1: TDBCheckBox;
  Bevel3: TBevel;

```

```

Bevel4: TBevel;
Bevel5: TBevel;
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton9Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public

```



```

    { Public declarations }
end;

var
    shoptolet: Tshoptolet;

implementation

    uses unit45, unit13, unit25;

    {$R *.dfm}

    procedure Tshoptolet.ToolButton1Click(Sender: TObject);
    begin
        dm.tkshop.Insert;
    end;

    procedure Tshoptolet.ToolButton3Click(Sender: TObject);
    begin
        dm.tkshop.Prior;
    end;

    procedure Tshoptolet.ToolButton5Click(Sender: TObject);
    begin
        dm.tkshop.Next;
    end;

    procedure Tshoptolet.ToolButton7Click(Sender: TObject);
    begin
        dm.tkshop.Cancel;
    end;

    procedure Tshoptolet.ToolButton9Click(Sender: TObject);
    begin
        try
            if(Application.MessageBox('Record will be deleted are you
            sure?','Confirmation',MB_YESNO)=IDYES) then
                dm.tkshop.Delete;
            except

                ShowMessage('Cant delete empty record!');
            end;
        end;

    procedure Tshoptolet.BitBtn1Click(Sender: TObject);
    begin
        shoptoletsearch.ShowModal;
    end;

    procedure Tshoptolet.BitBtn2Click(Sender: TObject);

```

```

begin
shoptoletreport.QuickRep1.Preview;
end;

procedure Tshoptolet.ToolButton11Click(Sender: TObject);
begin
dm.tkshop.Edit;
dm.tkshop.Post;
ShowMessage('Record is registered');
end;

procedure Tshoptolet.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tshoptolet.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Tshoptolet.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Tshoptolet.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already rented';
end;
if DBCheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='This shop give to let';
end;
end;

procedure Tshoptolet.FormCreate(Sender: TObject);
begin
shoptolet.ClientHeight:=614;
shoptolet.ClientWidth:=695;
end;

end.

```

unit Unit6;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,  
ToolWin;

type

Tplot = class(TForm)  
  ToolBar1: TToolBar;  
  ToolButton1: TToolButton;  
  ToolButton2: TToolButton;  
  ToolButton3: TToolButton;  
  ToolButton4: TToolButton;  
  ToolButton5: TToolButton;  
  ToolButton6: TToolButton;  
  ToolButton7: TToolButton;  
  ToolButton8: TToolButton;  
  ToolButton9: TToolButton;  
  ToolButton10: TToolButton;  
  ToolButton11: TToolButton;  
  ImageList1: TImageList;  
  Bevel1: TBevel;  
  Bevel2: TBevel;  
  BitBtn1: TBitBtn;  
  BitBtn2: TBitBtn;  
  DBCheckBox1: TDBCheckBox;  
  Bevel3: TBevel;  
  Bevel4: TBevel;  
  Bevel5: TBevel;  
  GroupBox1: TGroupBox;  
  GroupBox2: TGroupBox;  
  Label1: TLabel;  
  Label2: TLabel;  
  Label3: TLabel;  
  Label4: TLabel;  
  Label5: TLabel;  
  Label6: TLabel;  
  DBEdit1: TDBEdit;  
  DBEdit2: TDBEdit;  
  DBEdit3: TDBEdit;  
  DBEdit4: TDBEdit;  
  DBEdit5: TDBEdit;  
  DBEdit6: TDBEdit;  
  GroupBox3: TGroupBox;  
  Label7: TLabel;



```

Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  plot: Tplot;

implementation

  uses unit45, unit16, unit28;

{$R *.dfm}

procedure Tplot.ToolButton1Click(Sender: TObject);
begin
  dm.tsplot.Insert;
end;

procedure Tplot.ToolButton3Click(Sender: TObject);
begin
  dm.tsplot.Prior;
end;

procedure Tplot.ToolButton5Click(Sender: TObject);
begin

```

```

dm.tsplot.Next;
end;

procedure Tplot.ToolButton7Click(Sender: TObject);
begin
dm.tsplot.Cancel;
end;

procedure Tplot.ToolButton11Click(Sender: TObject);
begin
dm.tsplot.Edit;
dm.tsplot.Post;
ShowMessage('Record is registered');
end;

procedure Tplot.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already sold';
end;
if DBcheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell plot';
end;
end;

procedure Tplot.BitBtn1Click(Sender: TObject);
begin
plotforsalesearch.ShowModal;
end;

procedure Tplot.BitBtn2Click(Sender: TObject);
begin
plotforsalereport.QuickRep1.Preview;
end;

procedure Tplot.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tplot.ToolButton9Click(Sender: TObject);
begin
try

```

```

if (Application.MessageBox('Record will be deleted are you
sure?', 'Confirmation', MB_YESNO)=IDYES) then
dm.tsplot.Delete;
except
  ShowMessage('Cant delete empty record!');
end;

end;

procedure Tplot.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Tplot.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Tplot.FormCreate(Sender: TObject);
begin
plot.ClientHeight:=608;
plot.ClientWidth:=695;
end;

end.

unit Unit7;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,
  ToolWin;

type
  Tgarden = class(TForm)
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    ToolButton6: TToolButton;
    ToolButton7: TToolButton;
    ToolButton8: TToolButton;
    ToolButton9: TToolButton;
    ToolButton10: TToolButton;
    ToolButton11: TToolButton;

```



```

ImageList1: TImageList;
Bevel1: TBevel;
Bevel2: TBevel;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
DBCheckBox1: TDBCheckBox;
Bevel3: TBevel;
Bevel4: TBevel;
Bevel5: TBevel;
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
{ Private declarations }

```

```

public
  { Public declarations }
end;

var
  garden: Tgarden;

implementation

  uses unit45, unit17, unit29;

  {$R *.dfm}

  procedure Tgarden.DBEdit1Enter(Sender: TObject);
  begin
    if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
  end;

  procedure Tgarden.DBEdit1Exit(Sender: TObject);
  begin
    if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
  end;

  procedure Tgarden.ToolButton1Click(Sender: TObject);
  begin
    dm.ts花园.Insert;
  end;

  procedure Tgarden.ToolButton3Click(Sender: TObject);
  begin
    dm.ts花园.Prior;
  end;

  procedure Tgarden.ToolButton5Click(Sender: TObject);
  begin
    dm.ts花园.Next;
  end;

  procedure Tgarden.ToolButton7Click(Sender: TObject);
  begin
    dm.ts花园.Cancel;
  end;

  procedure Tgarden.ToolButton11Click(Sender: TObject);
  begin
    dm.ts花园.Edit;
    dm.ts花园.Post;
    ShowMessage('Record is registered');
  end;

  procedure Tgarden.BitBtn1Click(Sender: TObject);

```

```

begin
gardenforsalesearch.ShowModal;
end;

procedure Tgarden.BitBtn2Click(Sender: TObject);
begin
gardenforsalereport.QuickRep1.Preview;
end;

procedure Tgarden.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already sold';
end;
if DBcheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell garden';
end;
end;

procedure Tgarden.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tgarden.ToolButton9Click(Sender: TObject);
begin
try
if (Application.MessageBox('Record wii be deleted are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tsgarden.Delete;
except
ShowMessage('Cant delete empty record!');
end;
end;

procedure Tgarden.FormCreate(Sender: TObject);
begin
garden.ClientHeight:=617;
garden.ClientWidth:=695;
end;

end.

```



```
unit Unit8;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,  
ToolWin;
```

```
type
```

```
Tbuilding = class(TForm)  
  ToolBar1: TToolBar;  
  ToolButton1: TToolButton;  
  ToolButton2: TToolButton;  
  ToolButton3: TToolButton;  
  ToolButton4: TToolButton;  
  ToolButton5: TToolButton;  
  ToolButton6: TToolButton;  
  ToolButton7: TToolButton;  
  ToolButton8: TToolButton;  
  ToolButton9: TToolButton;  
  ToolButton10: TToolButton;  
  ToolButton11: TToolButton;  
  ImageList1: TImageList;  
  Bevel1: TBevel;  
  Bevel2: TBevel;  
  Bevel3: TBevel;  
  Bevel4: TBevel;  
  Bevel5: TBevel;  
  BitBtn1: TBitBtn;  
  BitBtn2: TBitBtn;  
  GroupBox1: TGroupBox;  
  GroupBox2: TGroupBox;  
  GroupBox3: TGroupBox;  
  Label1: TLabel;  
  Label2: TLabel;  
  Label3: TLabel;  
  Label4: TLabel;  
  Label5: TLabel;  
  Label6: TLabel;  
  Label7: TLabel;  
  Label8: TLabel;  
  Label9: TLabel;  
  Label10: TLabel;  
  Label11: TLabel;  
  Label12: TLabel;  
  Label13: TLabel;  
  Label14: TLabel;  
  Label15: TLabel;
```

```

DBCheckBox1: TDBCheckBox;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
DBEdit13: TDBEdit;
DBEdit14: TDBEdit;
DBEdit15: TDBEdit;
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  building: Tbuilding;

implementation

uses unit45, unit18, unit30;

{$R *.dfm}

procedure Tbuilding.DBEdit1Enter(Sender: TObject);
begin
  if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Tbuilding.DBEdit1Exit(Sender: TObject);
begin

```

```
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;  
end;
```

```
procedure Tbuilding.ToolButton1Click(Sender: TObject);  
begin  
dm.tsbuilding.Insert;  
end;
```

```
procedure Tbuilding.ToolButton3Click(Sender: TObject);  
begin  
dm.tsbuilding.Prior  
end;
```

```
procedure Tbuilding.ToolButton5Click(Sender: TObject);  
begin  
dm.tsbuilding.Next  
end;
```

```
procedure Tbuilding.ToolButton7Click(Sender: TObject);  
begin  
dm.tsbuilding.Cancel;  
end;
```

```
procedure Tbuilding.ToolButton11Click(Sender: TObject);  
begin  
dm.tsbuilding.Edit;  
dm.tsbuilding.Post;  
ShowMessage('Record is registered');  
end;
```

```
procedure Tbuilding.BitBtn1Click(Sender: TObject);  
begin  
buildingforsalesearch.ShowModal;  
end;
```

```
procedure Tbuilding.BitBtn2Click(Sender: TObject);  
begin  
buildingforsalereport.QuickRep1.Preview;  
end;
```

```
procedure Tbuilding.DBCheckBox1Click(Sender: TObject);  
begin  
if DBCheckBox1.Checked=true then  
begin  
GroupBox2.Visible:=true;  
DBCheckBox1.Caption:=' Already sold';  
end;  
if DBcheckbox1.Checked=false then  
begin  
GroupBox2.Visible:=false;  
DBCheckBox1.Caption:='Sell building';
```



```
end;  
end;
```

```
procedure Tbuilding.FormKeyPress(Sender: TObject; var Key: Char);  
begin  
  if (Key = #13) then  
  begin  
    key := #0;  
    Perform(WM_NEXTDLGCTL, 0, 0);  
  end;  
end;
```

```
procedure Tbuilding.ToolButton9Click(Sender: TObject);  
begin  
  try  
  if (Application.MessageBox('Record wii be deleted are you  
sure?', 'Confirmation', MB_YESNO)=IDYES) then  
dm.tsbuilding.Delete;  
  except  
    ShowMessage('Cant delete empty record!');  
  end;  
  
end;
```

```
procedure Tbuilding.FormCreate(Sender: TObject);  
begin  
  building.ClientHeight:=593;  
  building.ClientWidth:=695;  
end;  
  
end.
```

```
unit Unit9;
```

```
interface
```

```
uses  
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
  Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,  
  ToolWin;
```

```
type  
  Tfield = class(TForm)  
    ToolBar1: TToolBar;  
    ToolButton1: TToolButton;  
    ToolButton2: TToolButton;  
    ToolButton3: TToolButton;  
    ToolButton4: TToolButton;  
    ToolButton5: TToolButton;
```

```

ToolButton6: TToolButton;
ToolButton7: TToolButton;
ToolButton8: TToolButton;
ToolButton9: TToolButton;
ToolButton10: TToolButton;
ToolButton11: TToolButton;
ImageList1: TImageList;
Bevel1: TBevel;
Bevel2: TBevel;
Bevel3: TBevel;
Bevel4: TBevel;
Bevel5: TBevel;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
DBCheckBox1: TDBCheckBox;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
procedure ToolButton9Click(Sender: TObject);

```

```

    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure DBEdit1Enter(Sender: TObject);
    procedure DBEdit1Exit(Sender: TObject);
    procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    field: Tfield;

implementation

    uses unit45, unit20, unit31;

{$R *.dfm}

    procedure Tfield.ToolButton1Click(Sender: TObject);
    begin
        dm.tsfield.Insert;
    end;

    procedure Tfield.ToolButton3Click(Sender: TObject);
    begin
        dm.tsfield.Prior;
    end;

    procedure Tfield.ToolButton5Click(Sender: TObject);
    begin
        dm.tsfield.Next;
    end;

    procedure Tfield.ToolButton7Click(Sender: TObject);
    begin
        dm.tsfield.Cancel;
    end;

    procedure Tfield.ToolButton11Click(Sender: TObject);
    begin
        dm.tsfield.Edit;
        dm.tsfield.Post;
        ShowMessage('Record is registered');
    end;

    procedure Tfield.BitBtn1Click(Sender: TObject);
    begin
        fieldforsalesearch.ShowModal;
    end;

```



```

procedure Tfield.BitBtn2Click(Sender: TObject);
begin
fieldforsalereport.QuickRep1.Preview;
end;

procedure Tfield.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already sold';
end;
if DBcheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell field';
end;

end;

procedure Tfield.ToolButton9Click(Sender: TObject);
begin
try
if (Application.MessageBox('Record wii be deleted are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tsfield.Delete;
except
ShowMessage('Cant delete empty record!');
end;
end;

procedure Tfield.FormKeyPress(Sender: TObject; var Key: Char);
begin
If (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tfield.DBEdit1Enter(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

procedure Tfield.DBEdit1Exit(Sender: TObject);
begin
if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Tfield.FormCreate(Sender: TObject);

```

```
begin
field.ClientHeight:=607;
field.ClientWidth:=695;
```

```
end;
```

```
end.
```

```
unit Unit10;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, ImgList, ComCtrls,
ToolWin;
```

```
type
```

```
Tvilla = class(TForm)
  ToolBar1: TToolBar;
  ToolButton1: TToolButton;
  ToolButton2: TToolButton;
  ToolButton3: TToolButton;
  ToolButton4: TToolButton;
  ToolButton5: TToolButton;
  ToolButton6: TToolButton;
  ToolButton7: TToolButton;
  ToolButton8: TToolButton;
  ToolButton9: TToolButton;
  ToolButton10: TToolButton;
  ToolButton11: TToolButton;
  ImageList1: TImageList;
  Bevel1: TBevel;
  Bevel2: TBevel;
  Bevel3: TBevel;
  Bevel4: TBevel;
  Bevel5: TBevel;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  GroupBox1: TGroupBox;
  GroupBox2: TGroupBox;
  GroupBox3: TGroupBox;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  Label6: TLabel;
  Label7: TLabel;
```

```

Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
DBCheckBox1: TDBCheckBox;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure ToolButton9Click(Sender: TObject);
procedure DBCheckBox1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  villa: Tvilla;

implementation

  uses unit45, unit19, unit32;

{$R *.dfm}

procedure Tvilla.DBEdit1Enter(Sender: TObject);
begin
  if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
end;

```



```

procedure Tvilla.DBEdit1Exit(Sender: TObject);
begin
  if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
end;

procedure Tvilla.FormCreate(Sender: TObject);
begin
  villa.ClientHeight:=609;
  villa.ClientWidth:=695;
end;

procedure Tvilla.ToolButton1Click(Sender: TObject);
begin
  dm.tsvilla.Insert;
end;

procedure Tvilla.ToolButton3Click(Sender: TObject);
begin
  dm.tsvilla.Prior;
end;

procedure Tvilla.ToolButton5Click(Sender: TObject);
begin
  dm.tsvilla.Next;
end;

procedure Tvilla.ToolButton7Click(Sender: TObject);
begin
  dm.tsvilla.Cancel;
end;

procedure Tvilla.ToolButton11Click(Sender: TObject);
begin
  dm.tsvilla.Edit;
  dm.tsvilla.Post;
  ShowMessage('Record is registered');
end;

procedure Tvilla.BitBtn1Click(Sender: TObject);
begin
  villaforsalesearch.ShowModal;
end;

procedure Tvilla.BitBtn2Click(Sender: TObject);
begin
  villaforsalereport.QuickRep1.Preview;
end;

procedure Tvilla.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key = #13) then

```

```

begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tvilla.ToolButton9Click(Sender: TObject);
begin
try
if (Application.MessageBox('Record will be deleted are you
sure?','Confirmation',MB_YESNO)=IDYES) then
dm.tsvilla.Delete;
except
    ShowMessage('Cant delete empty record!');
end;

end;

procedure Tvilla.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already sold';
end;
if DBcheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='Sell villa';
end;
end;

end.

```

```

unit Unit11;

```

```

interface

```

```

uses

```

```

    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, DBCtrls, StdCtrls, Mask, Buttons, ExtCtrls, ImgList, ComCtrls,
    ToolWin;

```

```

type

```

```

    Tfarm = class(TForm)
        ToolBar1: TToolBar;
        ToolButton1: TToolButton;
        ToolButton2: TToolButton;
        ToolButton3: TToolButton;

```

```

ToolButton4: TToolButton;
ToolButton5: TToolButton;
ToolButton6: TToolButton;
ToolButton7: TToolButton;
ToolButton8: TToolButton;
ToolButton9: TToolButton;
ToolButton10: TToolButton;
ToolButton11: TToolButton;
ImageList1: TImageList;
Bevel1: TBevel;
Bevel2: TBevel;
Bevel3: TBevel;
Bevel4: TBevel;
Bevel5: TBevel;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
DBCheckBox1: TDBCheckBox;
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
DBEdit12: TDBEdit;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
procedure DBEdit1Enter(Sender: TObject);
procedure DBEdit1Exit(Sender: TObject);
procedure ToolButton1Click(Sender: TObject);
procedure ToolButton3Click(Sender: TObject);
procedure ToolButton5Click(Sender: TObject);
procedure ToolButton7Click(Sender: TObject);
procedure ToolButton11Click(Sender: TObject);

```



```

    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure DBCheckBox1Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure ToolButton9Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    farm: Tfarm;

implementation

    uses unit45, unit21, unit33;

{$R *.dfm}

    procedure Tfarm.DBEdit1Enter(Sender: TObject);
    begin
        if sender is tdbedit then tdbedit(sender).Color:=clMoneyGreen;
    end;

    procedure Tfarm.DBEdit1Exit(Sender: TObject);
    begin
        if sender is tdbedit then tdbedit(sender).Color:=clMenuBar;
    end;

    procedure Tfarm.ToolButton1Click(Sender: TObject);
    begin
        dm.tsfarm.Insert;
    end;

    procedure Tfarm.ToolButton3Click(Sender: TObject);
    begin
        dm.tsfarm.Prior;
    end;

    procedure Tfarm.ToolButton5Click(Sender: TObject);
    begin
        dm.tsfarm.Next;
    end;

    procedure Tfarm.ToolButton7Click(Sender: TObject);
    begin
        dm.tsfarm.Cancel;
    end;

```

```

procedure Tfarm.ToolButton11Click(Sender: TObject);
begin
dm.tsfarm.Edit;
dm.tsfarm.Post;
ShowMessage('Record is registered');
end;

procedure Tfarm.BitBtn1Click(Sender: TObject);
begin
farmforsalesearch.ShowModal;
end;

procedure Tfarm.BitBtn2Click(Sender: TObject);
begin
farmforsalereport.QuickRep1.Preview;
end;

procedure Tfarm.DBCheckBox1Click(Sender: TObject);
begin
if DBCheckBox1.Checked=true then
begin
GroupBox2.Visible:=true;
DBCheckBox1.Caption:='Already sold';
end;
if DBcheckbox1.Checked=false then
begin
GroupBox2.Visible:=false;
DBCheckBox1.Caption:='sell farm';
end;
end;

procedure Tfarm.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tfarm.ToolButton9Click(Sender: TObject);
begin
try
if (Application.MessageBox('Record will be deleted are you
sure?', 'Confirmation', MB_YESNO)=IDYES) then
dm.tsfarm.Delete;
except
ShowMessage('Cant delete empty record!');
end;
end;

```

```

procedure Tfarm.FormCreate(Sender: TObject);
begin
farm.ClientHeight:=607;
farm.ClientWidth:=695;
end;

end.

```

```

unit Unit12;

```

```

interface

```

```

uses

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, ExtCtrls, Buttons, DB, DBTables;

```

```

type

```

```

  Thousetoletsearch = class(TForm)
    GroupBox1: TGroupBox;
    Edit1: TEdit;
    Edit2: TEdit;
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Bevel1: TBevel;
    Edit3: TEdit;
    Edit4: TEdit;
    procedure Edit1Enter(Sender: TObject);
    procedure Edit1Exit(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure DBGrid1DbClick(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

```

```

var

```



housetoletsearch: Thousetoletsearch;

implementation

uses unit45, unit2;

{ \$R \*.dfm }

```
procedure Thousetoletsearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;
```

```
procedure Thousetoletsearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;
```

```
procedure Thousetoletsearch.Edit1Change(Sender: TObject);
begin
if Edit1.Text<>'' then begin
dm.tkhouse.Filtered:=true;
dm.tkhouse.Filter:='[Squaremeter]=' + Edit1.Text;
end
else
dm.tkhouse.Filtered:=false;
end;
```

```
procedure Thousetoletsearch.Edit2Change(Sender: TObject);
begin
if Edit2.Text<>'' then begin
dm.tkhouse.Filtered:=true;
dm.tkhouse.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
end
else
dm.tkhouse.Filtered:=false;

end;
```

```
procedure Thousetoletsearch.Edit3Change(Sender: TObject);
begin
if Edit3.Text<>'' then begin
dm.tkhouse.Filtered:=true;
dm.tkhouse.Filter:='[Price]=' + Edit3.Text;
end
else
dm.tkhouse.Filtered:=false;
end;
```

```
procedure Thousetoletsearch.Edit4Change(Sender: TObject);
```

```

begin
if Edit4.Text<>" then begin
dm.tkhouse.Filtered:=true;
dm.tkhouse.Filter:='[Heatingsystem]=' + #39 + Edit4.Text + '*' + #39;
end
else
dm.tkhouse.Filtered:=false;
end;

procedure Thousetoletsearch.DBGrid1DbClick(Sender: TObject);
begin
housetolet.Show;
housetoletsearch.Close;
end;

procedure Thousetoletsearch.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //Return null if not chr or space.
Beep; //inform user with a beep sound.
end;
end;

procedure Thousetoletsearch.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //Return null if not chr or space.
Beep; //inform user with a beep sound.
end;
end;

procedure Thousetoletsearch.BitBtn1Click(Sender: TObject);
begin
housetoletsearch.Close;
end;

procedure Thousetoletsearch.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

procedure Thousetoletsearch.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];

```

```
housetoletsearch.ClientHeight:=516;  
housetoletsearch.ClientWidth:=590;  
end;
```

```
end.
```

```
unit Unit13;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;
```

```
type
```

```
Tshoptoletsearch = class(TForm)  
    RESEARC: TGroupBox;  
    Label1: TLabel;  
    Label2: TLabel;  
    Label3: TLabel;  
    Label4: TLabel;  
    Edit1: TEdit;  
    Edit2: TEdit;  
    Edit3: TEdit;  
    Edit4: TEdit;  
    Bevel1: TBevel;  
    BitBtn1: TBitBtn;  
    DBGrid1: TDBGrid;  
    procedure Edit1Change(Sender: TObject);  
    procedure Edit2Change(Sender: TObject);  
    procedure Edit3Change(Sender: TObject);  
    procedure Edit4Change(Sender: TObject);  
    procedure Edit1Enter(Sender: TObject);  
    procedure Edit1Exit(Sender: TObject);  
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);  
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);  
    procedure DBGrid1DblClick(Sender: TObject);  
    procedure BitBtn1Click(Sender: TObject);  
    procedure FormKeyPress(Sender: TObject; var Key: Char);  
    procedure FormCreate(Sender: TObject);  
private  
    { Private declarations }  
public  
    { Public declarations }  
end;
```

```
var
```

```
shoptoletsearch: Tshoptoletsearch;
```



implementation

uses unit45, unit5;

{SR \*.dfm}

```
procedure Tshoptoletsearch.Edit1Change(Sender: TObject);
begin
if Edit1.Text<>'' then begin
dm.tkshop.Filtered:=true;
dm.tkshop.Filter:='[Squaremeter]=' + Edit1.Text;
end
else
dm.tkshop.Filtered:=false;
end;
```

```
procedure Tshoptoletsearch.Edit2Change(Sender: TObject);
begin
if Edit2.Text<>'' then begin
dm.tkshop.Filtered:=true;
dm.tkshop.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
end
else
dm.tkshop.Filtered:=false;
end;
```

```
procedure Tshoptoletsearch.Edit3Change(Sender: TObject);
begin
if Edit1.Text<>'' then begin
dm.tkshop.Filtered:=true;
dm.tkshop.Filter:='[Price]=' + Edit3.Text;
end
else
dm.tkshop.Filtered:=false;
end;
```

```
procedure Tshoptoletsearch.Edit4Change(Sender: TObject);
begin
if Edit4.Text<>'' then begin
dm.tkshop.Filtered:=true;
dm.tkshop.Filter:='[Heatingsystem]=' + #39 + Edit4.Text + '*' + #39;
end
else
dm.tkshop.Filtered:=false;
end;
```

```
procedure Tshoptoletsearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;
```

```

procedure Tshoptoletsearch.Edit1Exit(Sender: TObject);
begin
  if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

procedure Tshoptoletsearch.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if not (key in ['0'..'9',#8,#13]) then
  begin
    key:=#0; //return null if not chr or a space.
    Beep;    //inform user with a beep sound.
  end;
end;

procedure Tshoptoletsearch.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
  if not (key in ['0'..'9',#8,#13]) then
  begin
    key:=#0; //return null if not chr or a space.
    Beep;    //inform user with a beep sound.
  end;
end;

procedure Tshoptoletsearch.DBGrid1DbClick(Sender: TObject);
begin
  shoptolet.Show;
  shoptoletsearch.Close;
end;

procedure Tshoptoletsearch.BitBtn1Click(Sender: TObject);
begin
  shoptoletsearch.Close;
end;

procedure Tshoptoletsearch.FormKeyPress(Sender: TObject; var Key: Char);
begin
  If (Key = #13) then
  begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
  end;
end;

procedure Tshoptoletsearch.FormCreate(Sender: TObject);
begin
  borderIcons:=borderIcons-[bistystemmenu];
  shoptoletsearch.ClientHeight:=516;
  shoptoletsearch.ClientWidth:=595;
end;

```

end.

unit Unit14;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

type

Thouseforsalesearch = class(TForm)

GroupBox1: TGroupBox;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Bevel1: TBevel;

BitBtn1: TBitBtn;

DBGrid1: TDBGrid;

Edit4: TEdit;

Edit3: TEdit;

Edit2: TEdit;

Edit1: TEdit;

procedure Edit1Change(Sender: TObject);

procedure Edit1Enter(Sender: TObject);

procedure Edit1Exit(Sender: TObject);

procedure Edit2Change(Sender: TObject);

procedure Edit3Change(Sender: TObject);

procedure Edit4Change(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure DBGrid1DbClick(Sender: TObject);

procedure Edit1KeyPress(Sender: TObject; var Key: Char);

procedure Edit3KeyPress(Sender: TObject; var Key: Char);

procedure FormKeyPress(Sender: TObject; var Key: Char);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

houseforsalesearch: Thouseforsalesearch;

implementation

uses unit45, unit4;

{ \$R \*.dfm }



```

procedure Thouseforsalesearch.Edit1Change(Sender: TObject);
begin
if Edit1.Text<>'' then begin
dm.tshouse.Filtered:=true;
dm.tshouse.Filter:='[Squaremeter]=' + Edit1.Text;
end
else
dm.tshouse.Filtered:=false;
end;

```

```

procedure Thouseforsalesearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

```

```

procedure Thouseforsalesearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

```

```

procedure Thouseforsalesearch.Edit2Change(Sender: TObject);
begin
if Edit2.Text<>'' then begin
dm.tshouse.Filtered:=true;
dm.tshouse.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
end
else
dm.tshouse.Filtered:=false;
end;

```

```

procedure Thouseforsalesearch.Edit3Change(Sender: TObject);
begin
if Edit3.Text<>'' then begin
dm.tshouse.Filtered:=true;
dm.tshouse.Filter:='[Price]=' + Edit3.Text;
end
else
dm.tshouse.Filtered:=false;
end;

```

```

procedure Thouseforsalesearch.Edit4Change(Sender: TObject);
begin
if Edit4.Text<>'' then begin
dm.tshouse.Filtered:=true;
dm.tshouse.Filter:='[Heatingsystem]=' + #39 + Edit4.Text + '*' + #39;
end
else
dm.tshouse.Filtered:=false;
end;

```

```

procedure Thouseforsalesearch.BitBtn1Click(Sender: TObject);
begin
houseforsalesearch.Close;
end;

procedure Thouseforsalesearch.DBGrid1DbClick(Sender: TObject);
begin
houseforsale.Show;
houseforsalesearch.Close;
end;

procedure Thouseforsalesearch.Edit1KeyPress(Sender: TObject;
  var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep;    //inform user with e beep sound.
end;
end;

procedure Thouseforsalesearch.Edit3KeyPress(Sender: TObject;
  var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep;    //inform user with e beep sound.
end;
end;

procedure Thouseforsalesearch.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

procedure Thouseforsalesearch.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
houseforsalesearch.ClientHeight:=516;
houseforsalesearch.ClientWidth:=594;
end;

end.

```

```

unit Unit15;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

type
  Tshopforsalesearch = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Bevel1: TBevel;
    BitBtn1: TBitBtn;
    DBGrid1: TDBGrid;
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure Edit1Enter(Sender: TObject);
    procedure Edit1Exit(Sender: TObject);
    procedure DBGrid1DblClick(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  shopforsalesearch: Tshopforsalesearch;

implementation

uses unit45, unit3;

{$R *.dfm}

procedure Tshopforsalesearch.Edit1Change(Sender: TObject);

```

```

begin
if Edit1.Text<>" then begin
dm.tsshop.Filtered:=true;
dm.tsshop.Filter:='[Squaremeter]=' + Edit1.Text;
end
else
dm.tsshop.Filtered:=false;
end;

procedure Tshopforsalesearch.Edit2Change(Sender: TObject);
begin
if Edit2.Text<>" then begin
dm.tsshop.Filtered:=true;
dm.tsshop.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
end
else
dm.tsshop.Filtered:=false;
end;

procedure Tshopforsalesearch.Edit3Change(Sender: TObject);
begin
if Edit1.Text<>" then begin
dm.tsshop.Filtered:=true;
dm.tsshop.Filter:='[Price]=' + Edit3.Text;
end
else
dm.tsshop.Filtered:=false;
end;

procedure Tshopforsalesearch.Edit4Change(Sender: TObject);
begin
if Edit4.Text<>" then begin
dm.tsshop.Filtered:=true;
dm.tsshop.Filter:='[Heatingsystem]=' + #39 + Edit4.Text + '*' + #39;
end
else
dm.tsshop.Filtered:=false;
end;

procedure Tshopforsalesearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

procedure Tshopforsalesearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

procedure Tshopforsalesearch.DBGrid1DbClick(Sender: TObject);
begin

```



```
shopforsale.Show;
shopforsalesearch.Close;
end;
```

```
procedure Tshopforsalesearch.BitBtn1Click(Sender: TObject);
begin
shopforsalesearch.Close;
end;
```

```
procedure Tshopforsalesearch.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or a space.
Beep; //inform user with a beep sound.
end;
end;
```

```
procedure Tshopforsalesearch.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or a space.
Beep; //inform user with a beep sound.
end;
end;
```

```
procedure Tshopforsalesearch.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;
```

```
end;
```

```
procedure Tshopforsalesearch.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
shopforsalesearch.ClientHeight:=516;
shopforsalesearch.ClientWidth:=597;
end;
```

```
end.
```

```
unit Unit16;
```

```
interface
```

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

type

```
Tplotforsalesearch = class(TForm)
  GroupBox1: TGroupBox;
  Bevel1: TBevel;
  BitBtn1: TBitBtn;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  DBGrid1: TDBGrid;
  Edit1: TEdit;
  Edit2: TEdit;
  Edit3: TEdit;
  procedure Edit1Change(Sender: TObject);
  procedure Edit2Change(Sender: TObject);
  procedure Edit3Change(Sender: TObject);
  procedure Edit1Enter(Sender: TObject);
  procedure Edit1Exit(Sender: TObject);
  procedure DBGrid1DbClick(Sender: TObject);
  procedure BitBtn1Click(Sender: TObject);
  procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  procedure Edit3KeyPress(Sender: TObject; var Key: Char);
  procedure FormKeyPress(Sender: TObject; var Key: Char);
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

var

plotforsalesearch: Tplotforsalesearch;

implementation

uses unit45, unit6;

{ \$R \*.dfm }

```
procedure Tplotforsalesearch.Edit1Change(Sender: TObject);
begin
  if Edit1.Text<>'' then begin
    dm.tsplot.Filtered:=true;
    dm.tsplot.Filter:='[Squaremeter]=' + edit1.Text;
  end
  else
    dm.tsplot.Filtered:=false;
```

```

end;

procedure Tplotforsalesearch.Edit2Change(Sender: TObject);
begin
if Edit2.Text<>'' then begin
dm.tsplot.Filtered:=true;
dm.tsplot.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
end
else
dm.tsplot.Filtered:=false;
end;

procedure Tplotforsalesearch.Edit3Change(Sender: TObject);
begin
if Edit3.Text<>'' then begin
dm.tsplot.Filtered:=true;
dm.tsplot.Filter:='[Price]=' + Edit3.Text;
end
else
dm.tsplot.Filtered:=false;
end;

procedure Tplotforsalesearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

procedure Tplotforsalesearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

procedure Tplotforsalesearch.DBGrid1DbClick(Sender: TObject);
begin
plot.Show;
plotforsalesearch.Close;
end;

procedure Tplotforsalesearch.BitBtn1Click(Sender: TObject);
begin
plotforsalesearch.Close;
end;

procedure Tplotforsalesearch.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or a space.
Beep; //inform user with a beep sound.
end;
end;

```

```

procedure Tplotforsalesearch.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
  if not (key in ['0'..'9',#8,#13]) then
    begin
      key:=#0; //return null if not chr or a space.
      Beep;    //inform user with a beep sound.
    end;
  end;
end;

```

```

procedure Tplotforsalesearch.FormKeyPress(Sender: TObject; var Key: Char);
begin
  If (Key = #13) then
    begin
      key := #0;
      Perform(WM_NEXTDLGCTL, 0, 0);
    end;
end;

```

```

end;

```

```

procedure Tplotforsalesearch.FormCreate(Sender: TObject);
begin
  borderIcons:=borderIcons-[bisystemmenu];
  plotforsalesearch.ClientHeight:=516;
  plotforsalesearch.ClientWidth:=595;
end;

```

```

end.

```

```

unit Unit17;

```

```

interface

```

```

uses

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

```

```

type

```

```

  Tgardenforsalesearch = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Bevel1: TBevel;
    BitBtn1: TBitBtn;
    DBGrid1: TDBGrid;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;

```



```

    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure DBGrid1DbClick(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Edit1Enter(Sender: TObject);
    procedure Edit1Exit(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    gardenforsalesearch: Tgardenforsalesearch;

implementation

    uses unit45, unit7;

{$R *.dfm}

    procedure Tgardenforsalesearch.Edit1Change(Sender: TObject);
    begin
        if Edit1.Text<>'' then begin
            dm.tsgarden.Filtered:=true;
            dm.tsgarden.Filter:='[Squaremeter]=' + edit1.Text;
        end
        else
            dm.tsgarden.Filtered:=false;
        end;

    procedure Tgardenforsalesearch.Edit2Change(Sender: TObject);
    begin
        if Edit2.Text<>'' then begin
            dm.tsgarden.Filtered:=true;
            dm.tsgarden.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
        end
        else
            dm.tsgarden.Filtered:=false;
        end;

    procedure Tgardenforsalesearch.Edit3Change(Sender: TObject);
    begin
        if Edit3.Text<>'' then begin
            dm.tsgarden.Filtered:=true;
            dm.tsgarden.Filter:='[Price]=' + Edit3.Text;

```

```

end
else
dm.tsgarden.Filtered:=false;
end;

procedure Tgardenforsalesearch.DBGrid1DbClick(Sender: TObject);
begin
garden.Show;
gardenforsalesearch.Close;
end;

procedure Tgardenforsalesearch.BitBtn1Click(Sender: TObject);
begin
gardenforsalesearch.Close;
end;

procedure Tgardenforsalesearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

procedure Tgardenforsalesearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

procedure Tgardenforsalesearch.Edit1KeyPress(Sender: TObject;
var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep; //inform user with e beep sound.
end;
end;

procedure Tgardenforsalesearch.Edit3KeyPress(Sender: TObject;
var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep; //inform user with e beep sound.
end;
end;

procedure Tgardenforsalesearch.FormKeyPress(Sender: TObject;
var Key: Char);
begin
if (Key = #13) then

```

```

begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

procedure Tgardenforsalesearch.FormCreate(Sender: TObject);
begin
    borderIcons:=borderIcons-[bisystemmenu];
    gardenforsalesearch.ClientHeight:=514;
    gardenforsalesearch.ClientWidth:=585;
end;

end.

unit Unit18;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

type
    Tbuildingforsalesearch = class(TForm)
        GroupBox1: TGroupBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Bevel1: TBevel;
        BitBtn1: TBitBtn;
        DBGrid1: TDBGrid;
        Edit1: TEdit;
        Edit2: TEdit;
        Edit3: TEdit;
        Edit4: TEdit;
        procedure Edit1Change(Sender: TObject);
        procedure Edit2Change(Sender: TObject);
        procedure Edit3Change(Sender: TObject);
        procedure Edit4Change(Sender: TObject);
        procedure DBGrid1DblClick(Sender: TObject);
        procedure BitBtn1Click(Sender: TObject);
        procedure Edit1KeyPress(Sender: TObject; var Key: Char);
        procedure Edit3KeyPress(Sender: TObject; var Key: Char);
        procedure Edit1Enter(Sender: TObject);
        procedure Edit1Exit(Sender: TObject);
        procedure FormKeyPress(Sender: TObject; var Key: Char);
        procedure FormCreate(Sender: TObject);
    end;

```

```

private
  { Private declarations }
public
  { Public declarations }
end;

var
  buildingforsalesearch: Tbuildingforsalesearch;

implementation

  uses unit45, unit8;

{$R *.dfm}

procedure Tbuildingforsalesearch.Edit1Change(Sender: TObject);
begin
  if Edit1.Text<>'' then begin
    dm.tsbuilding.Filtered:=true;
    dm.tsbuilding.Filter:='[Squaremeter]=' + Edit1.Text;
  end
  else
    dm.tsbuilding.Filtered:=false;
  end;

procedure Tbuildingforsalesearch.Edit2Change(Sender: TObject);
begin
  if Edit2.Text<>'' then begin
    dm.tsbuilding.Filtered:=true;
    dm.tsbuilding.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
  end
  else
    dm.tsbuilding.Filtered:=false;
  end;

end;

procedure Tbuildingforsalesearch.Edit3Change(Sender: TObject);
begin
  if edit3.Text<>'' then begin
    dm.tsbuilding.Filtered:=true;
    dm.tsbuilding.Filter:='[Price]=' + Edit3.Text;
  end
  else
    dm.tsbuilding.Filtered:=false;
  end;

procedure Tbuildingforsalesearch.Edit4Change(Sender: TObject);
begin
  if edit4.Text<>'' then begin
    dm.tsbuilding.Filtered:=true;

```



```

dm.tsbuilding.Filter:=[Heatingsystem]=' + #39 + edit4.Text + '*' + #39;
end
else
dm.tsbuilding.Filtered:=false;
end;

```

```

procedure Tbuildingforsalesearch.DBGrid1DbClick(Sender: TObject);
begin
building.Show;
buildingforsalesearch.Close;
end;

```

```

procedure Tbuildingforsalesearch.BitBtn1Click(Sender: TObject);
begin
buildingforsalesearch.Close;
end;

```

```

procedure Tbuildingforsalesearch.Edit1KeyPress(Sender: TObject;
var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep; //inform user with e beep sound.
end;
end;

```

```

procedure Tbuildingforsalesearch.Edit3KeyPress(Sender: TObject;
var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep; //inform user with e beep sound.
end;
end;

```

```

procedure Tbuildingforsalesearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

```

```

procedure Tbuildingforsalesearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

```

```

procedure Tbuildingforsalesearch.FormKeyPress(Sender: TObject;
var Key: Char);
begin
if (Key = #13) then

```

```

begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

procedure Tbuildingforsalesearch.FormCreate(Sender: TObject);
begin
    borderIcons:=borderIcons-[bisystemmenu];
    buildingforsalesearch.ClientHeight:=516;
    buildingforsalesearch.ClientWidth:=589;
end;

end.

unit Unit19;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

type
    Tvillaforsalesearch = class(TForm)
        GroupBox1: TGroupBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Edit1: TEdit;
        Edit2: TEdit;
        Edit3: TEdit;
        Bevel1: TBevel;
        BitBtn1: TBitBtn;
        DBGrid1: TDBGrid;
        procedure Edit1Change(Sender: TObject);
        procedure Edit2Change(Sender: TObject);
        procedure Edit3Change(Sender: TObject);
        procedure Edit1Enter(Sender: TObject);
        procedure Edit1Exit(Sender: TObject);
        procedure DBGrid1DblClick(Sender: TObject);
        procedure BitBtn1Click(Sender: TObject);
        procedure Edit1KeyPress(Sender: TObject; var Key: Char);
        procedure Edit3KeyPress(Sender: TObject; var Key: Char);
        procedure FormKeyPress(Sender: TObject; var Key: Char);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public

```

```

    { Public declarations }
end;

var
    villaforsalesearch: Tvillaforsalesearch;

implementation

    uses unit45, unit10;

{$R *.dfm}

procedure Tvillaforsalesearch.Edit1Change(Sender: TObject);
begin
    if Edit1.Text<>'' then begin
        dm.tsvilla.Filtered:=true;
        dm.tsvilla.Filter:='[Squaremeter]=' + Edit1.Text;
    end
    else
        dm.tsvilla.Filtered:=false;
    end;

procedure Tvillaforsalesearch.Edit2Change(Sender: TObject);
begin
    if Edit2.Text<>'' then begin
        dm.tsvilla.Filtered:=true;
        dm.tsvilla.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
    end
    else
        dm.tsvilla.Filtered:=false;
    end;

procedure Tvillaforsalesearch.Edit3Change(Sender: TObject);
begin
    if Edit3.Text<>'' then begin
        dm.tsvilla.Filtered:=true;
        dm.tsvilla.Filter:='[Price]=' + Edit3.Text;
    end
    else
        dm.tsvilla.Filtered:=false;
    end;

procedure Tvillaforsalesearch.Edit1Enter(Sender: TObject);
begin
    if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

procedure Tvillaforsalesearch.Edit1Exit(Sender: TObject);
begin
    if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

```

```

procedure Tvillaforsalesearch.DBGrid1DbClick(Sender: TObject);
begin
villa.Show;
villaforsalesearch.Close;
end;

procedure Tvillaforsalesearch.BitBtn1Click(Sender: TObject);
begin
villaforsalesearch.Close;
end;

procedure Tvillaforsalesearch.Edit1KeyPress(Sender: TObject;
  var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or a space.
Beep;    //inform user with a beep sound.
end;
end;

procedure Tvillaforsalesearch.Edit3KeyPress(Sender: TObject;
  var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or a space.
Beep;    //inform user with a beep sound.
end;
end;

procedure Tvillaforsalesearch.FormKeyPress(Sender: TObject; var Key: Char);
begin
If (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

procedure Tvillaforsalesearch.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
villaforsalesearch.ClientHeight:=516;
villaforsalesearch.ClientWidth:=595;
end;

end.

```



```

unit Unit20;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

type
  Tfieldforsalesearch = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Bevel1: TBevel;
    BitBtn1: TBitBtn;
    DBGrid1: TDBGrid;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit1Enter(Sender: TObject);
    procedure Edit1Exit(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Edit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBGrid1DblClick(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  fieldforsalesearch: Tfieldforsalesearch;

implementation

uses unit45, unit9;

{$R *.dfm}

procedure Tfieldforsalesearch.Edit1Change(Sender: TObject);
begin
  if Edit1.Text<>'' then begin
    dm.tsfield.Filtered:=true;
  end;
end;

```

```

dm.tsfield.Filter:='[Squaremeter]=' + edit1.Text;
end
else
dm.tsfield.Filtered:=false;
end;

procedure Tfieldforsalesearch.Edit2Change(Sender: TObject);
begin
if Edit2.Text<>'' then begin
dm.tsfield.Filtered:=true;
dm.tsfield.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
end
else
dm.tsfield.Filtered:=false;
end;

procedure Tfieldforsalesearch.Edit3Change(Sender: TObject);
begin
if Edit3.Text<>'' then begin
dm.tsfield.Filtered:=true;
dm.tsfield.Filter:='[Price]=' + Edit3.Text;
end
else
dm.tsfield.Filtered:=false;
end;

procedure Tfieldforsalesearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

procedure Tfieldforsalesearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

procedure Tfieldforsalesearch.Edit1KeyPress(Sender: TObject;
var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep; //inform user with e beep sound.
end;
end;

procedure Tfieldforsalesearch.Edit3KeyPress(Sender: TObject;
var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin

```

```

key:=#0; //return null if not chr or space.
Beep;    //inform user with e beep sound.
end;
end;

procedure Tfieldforsalesearch.DBGrid1DbClick(Sender: TObject);
begin
field.Show;
fieldforsalesearch.Close;
end;

procedure Tfieldforsalesearch.BitBtn1Click(Sender: TObject);
begin
fieldforsalesearch.Close;
end;

procedure Tfieldforsalesearch.FormKeyPress(Sender: TObject; var Key: Char);
begin
If (Key = #13) then
begin
key := #0;
Perform(WM_NEXTDLGCTL, 0, 0);
end;

end;

end;

procedure Tfieldforsalesearch.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
fieldforsalesearch.ClientHeight:=516;
fieldforsalesearch.ClientWidth:=595;
end;

end.

unit Unit21;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Grids, DBGrids, StdCtrls, Buttons, ExtCtrls;

type
Tfarmforsalesearch = class(TForm)
GroupBox1: TGroupBox;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Bevel1: TBevel;

```

```

BitBtn1: TBitBtn;
DBGrid1: TDBGrid;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
procedure Edit1Change(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure Edit3Change(Sender: TObject);
procedure Edit1Enter(Sender: TObject);
procedure Edit1Exit(Sender: TObject);
procedure Edit1KeyPress(Sender: TObject; var Key: Char);
procedure Edit3KeyPress(Sender: TObject; var Key: Char);
procedure DBGrid1DblClick(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  farmforsalesearch: Tfarmforsalesearch;

implementation

uses unit45, unit11;

{$R *.dfm}

procedure Tfarmforsalesearch.Edit1Change(Sender: TObject);
begin
  if Edit1.Text<>'' then begin
    dm.tsfarm.Filtered:=true;
    dm.tsfarm.Filter:='[Squaremeter]=' + edit1.Text;
  end
  else
    dm.tsfarm.Filtered:=false;
end;

procedure Tfarmforsalesearch.Edit2Change(Sender: TObject);
begin
  if Edit2.Text<>'' then begin
    dm.tsfarm.Filtered:=true;
    dm.tsfarm.Filter:='[District]=' + #39 + Edit2.Text + '*' + #39;
  end
  else
    dm.tsfarm.Filtered:=false;
end;

```



```

procedure Tfarmforsalesearch.Edit3Change(Sender: TObject);
begin
if Edit3.Text<>'' then begin
dm.tsfarm.Filtered:=true;
dm.tsfarm.Filter:='[Price]=' + Edit3.Text;
end
else
dm.tsfarm.Filtered:=false;
end;

procedure Tfarmforsalesearch.Edit1Enter(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;

procedure Tfarmforsalesearch.Edit1Exit(Sender: TObject);
begin
if sender is tedit then tedit(sender).Color:=clMenuBar;
end;

procedure Tfarmforsalesearch.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep; //inform user with e beep sound.
end;
end;

procedure Tfarmforsalesearch.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8,#13]) then
begin
key:=#0; //return null if not chr or space.
Beep; //inform user with e beep sound.
end;
end;

procedure Tfarmforsalesearch.DBGrid1DbClick(Sender: TObject);
begin
farm.Show;
farmforsalesearch.Close;
end;

procedure Tfarmforsalesearch.BitBtn1Click(Sender: TObject);
begin
farmforsalesearch.Close;
end;

procedure Tfarmforsalesearch.FormKeyPress(Sender: TObject; var Key: Char);
begin

```

```

If (Key = #13) then
begin
    key := #0;
    Perform(WM_NEXTDLGCTL, 0, 0);
end;
end;

procedure Tfarmforsalesearch.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
farmforsalesearch.ClientHeight:=516;
farmforsalesearch.ClientWidth:=595;

end;

end.

unit Unit22;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, jpeg, ExtCtrls;

type
    Tabout = class(TForm)
        Image1: TImage;
        Label1: TLabel;
        Label2: TLabel;
        procedure FormCreate(Sender: TObject);
        procedure Image1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    about: Tabout;

implementation

{$R *.dfm}

procedure Tabout.FormCreate(Sender: TObject);
begin
    about.ClientHeight:=275;
    about.ClientWidth:=427;
end;

```

```

procedure Tabout.Image1Click(Sender: TObject);
begin

end;

end.

unit Unit23;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons;

type
  Tflierprint = class(TForm)
    Label1: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn7: TBitBtn;
    BitBtn8: TBitBtn;
    BitBtn9: TBitBtn;
    BitBtn10: TBitBtn;
    BitBtn11: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn7Click(Sender: TObject);
    procedure BitBtn8Click(Sender: TObject);
    procedure BitBtn9Click(Sender: TObject);
    procedure BitBtn10Click(Sender: TObject);
    procedure BitBtn11Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  flierprint: Tflierprint;

```

implementation

uses unit34, unit35, unit36, unit37, unit38, unit39, unit40, unit41, unit42, unit43;

{ \$R \*.dfm }

```
procedure Tflierprint.BitBtn1Click(Sender: TObject);
begin
form34.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn2Click(Sender: TObject);
begin
form35.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn3Click(Sender: TObject);
begin
form36.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn4Click(Sender: TObject);
begin
form37.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn5Click(Sender: TObject);
begin
form40.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn6Click(Sender: TObject);
begin
form43.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn7Click(Sender: TObject);
begin
form38.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn8Click(Sender: TObject);
begin
form42.QuickRep1.Preview;
end;
```

```
procedure Tflierprint.BitBtn9Click(Sender: TObject);
begin
form39.QuickRep1.Preview;
end;
```



```

procedure Tflierprint.BitBtn10Click(Sender: TObject);
begin
form41.QuickRep1.Preview;
end;

```

```

procedure Tflierprint.BitBtn11Click(Sender: TObject);
begin
flierprint.Close;
end;

```

```

procedure Tflierprint.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[biSystemMenu];
flierprint.ClientHeight:=599;
flierprint.ClientWidth:=611;
end;

```

```

end.

```

```

unit Unit23;

```

```

interface

```

```

uses

```

```

    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Buttons;

```

```

type

```

```

    Tflierprint = class(TForm)

```

```

        Label1: TLabel;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        BitBtn3: TBitBtn;
        BitBtn4: TBitBtn;
        BitBtn5: TBitBtn;
        BitBtn6: TBitBtn;
        BitBtn7: TBitBtn;
        BitBtn8: TBitBtn;
        BitBtn9: TBitBtn;
        BitBtn10: TBitBtn;
        BitBtn11: TBitBtn;

```

```

        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure BitBtn3Click(Sender: TObject);
        procedure BitBtn4Click(Sender: TObject);
        procedure BitBtn5Click(Sender: TObject);
        procedure BitBtn6Click(Sender: TObject);
        procedure BitBtn7Click(Sender: TObject);
        procedure BitBtn8Click(Sender: TObject);

```

```

    procedure BitBtn9Click(Sender: TObject);
    procedure BitBtn10Click(Sender: TObject);
    procedure BitBtn11Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    flierprint: Tflierprint;

implementation

uses unit34, unit35, unit36, unit37, unit38, unit39, unit40, unit41, unit42, unit43;

{$R *.dfm}

procedure Tflierprint.BitBtn1Click(Sender: TObject);
begin
    form34.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn2Click(Sender: TObject);
begin
    form35.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn3Click(Sender: TObject);
begin
    form36.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn4Click(Sender: TObject);
begin
    form37.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn5Click(Sender: TObject);
begin
    form40.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn6Click(Sender: TObject);
begin
    form43.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn7Click(Sender: TObject);
begin

```

```

form38.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn8Click(Sender: TObject);
begin
form42.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn9Click(Sender: TObject);
begin
form39.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn10Click(Sender: TObject);
begin
form41.QuickRep1.Preview;
end;

procedure Tflierprint.BitBtn11Click(Sender: TObject);
begin
flierprint.Close;
end;

procedure Tflierprint.FormCreate(Sender: TObject);
begin
borderIcons:=borderIcons-[bisystemmenu];
flierprint.ClientHeight:=599;
flierprint.ClientWidth:=611;
end;

end.

unit Unit25;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  Tshoptoletreport = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRLabel4: TQRLabel;

```

```

    QRLabel5: TQRLabel;
    QRLabel6: TQRLabel;
    QRLabel7: TQRLabel;
    QRLabel8: TQRLabel;
    QRLabel9: TQRLabel;
    QRLabel10: TQRLabel;
    QRLabel11: TQRLabel;
    QRLabel12: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRDBText4: TQRDBText;
    QRDBText5: TQRDBText;
    QRDBText6: TQRDBText;
    QRDBText7: TQRDBText;
    procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    shoptoletreport: Tshoptoletreport;

implementation

    uses unit45;

    {$R *.dfm}

    procedure Tshoptoletreport.FormCreate(Sender: TObject);
    begin

    end;

    end.

    unit Unit26;

    interface

    uses
        Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
        Dialogs, QRCtrls, QuickRpt, ExtCtrls;

    type
        Thouseforsalereport = class(TForm)
            QuickRep1: TQuickRep;
            PageHeaderBand1: TQRBand;
            DetailBand1: TQRBand;

```



```

QRDBText1: TQRDBText;
QRLabel1: TQRLabel;
QRLabel2: TQRLabel;
QRLabel3: TQRLabel;
QRLabel4: TQRLabel;
QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRLabel8: TQRLabel;
QRLabel9: TQRLabel;
QRLabel10: TQRLabel;
QRLabel11: TQRLabel;
QRLabel12: TQRLabel;
QRDBText2: TQRDBText;
QRDBText3: TQRDBText;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  houseforsalereport: Thouseforsalereport;

implementation

uses unit45;

{$R *.dfm}

procedure Thouseforsalereport.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit27;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

```

```

type
  Tshopforsalereport = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    QRLabel6: TQRLabel;
    QRLabel7: TQRLabel;
    QRLabel8: TQRLabel;
    QRLabel9: TQRLabel;
    QRLabel10: TQRLabel;
    QRLabel11: TQRLabel;
    QRLabel12: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRDBText4: TQRDBText;
    QRDBText5: TQRDBText;
    QRDBText6: TQRDBText;
    QRDBText7: TQRDBText;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  shopforsalereport: Tshopforsalereport;

implementation

  uses unit45;

  {$R *.dfm}

  procedure Tshopforsalereport.FormCreate(Sender: TObject);
  begin

  end;

end.

unit Unit28;

interface

```

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type

```
Tplotforsalereport = class(TForm)
  QuickRep1: TQuickRep;
  PageHeaderBand1: TQRBand;
  DetailBand1: TQRBand;
  QRDBText1: TQRDBText;
  QRLabel1: TQRLabel;
  QRLabel2: TQRLabel;
  QRLabel3: TQRLabel;
  QRLabel4: TQRLabel;
  QRLabel5: TQRLabel;
  QRLabel6: TQRLabel;
  QRLabel7: TQRLabel;
  QRLabel8: TQRLabel;
  QRLabel9: TQRLabel;
  QRLabel10: TQRLabel;
  QRLabel11: TQRLabel;
  QRLabel12: TQRLabel;
  QRDBText2: TQRDBText;
  QRDBText3: TQRDBText;
  QRDBText4: TQRDBText;
  QRDBText5: TQRDBText;
  QRDBText6: TQRDBText;
  QRDBText7: TQRDBText;
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

var

plotforsalereport: Tplotforsalereport;

implementation

uses unit45;

{ \$R \*.dfm }

```
procedure Tplotforsalereport.FormCreate(Sender: TObject);
begin
```

```
end;
```

```
end.
```

```
unit Unit29;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, QRCtrls, QuickRpt, ExtCtrls;
```

```
type
```

```
Tgardenforsalereport = class(TForm)
```

```
QuickRep1: TQuickRep;
```

```
PageHeaderBand1: TQRBand;
```

```
DetailBand1: TQRBand;
```

```
QRDBText1: TQRDBText;
```

```
QRLabel1: TQRLabel;
```

```
QRLabel2: TQRLabel;
```

```
QRLabel3: TQRLabel;
```

```
QRLabel4: TQRLabel;
```

```
QRLabel5: TQRLabel;
```

```
QRLabel6: TQRLabel;
```

```
QRLabel7: TQRLabel;
```

```
QRLabel8: TQRLabel;
```

```
QRLabel9: TQRLabel;
```

```
QRLabel10: TQRLabel;
```

```
QRLabel11: TQRLabel;
```

```
QRLabel12: TQRLabel;
```

```
QRDBText2: TQRDBText;
```

```
QRDBText3: TQRDBText;
```

```
QRDBText4: TQRDBText;
```

```
QRDBText5: TQRDBText;
```

```
QRDBText6: TQRDBText;
```

```
QRDBText7: TQRDBText;
```

```
procedure FormCreate(Sender: TObject);
```

```
private
```

```
{ Private declarations }
```

```
public
```

```
{ Public declarations }
```

```
end;
```

```
var
```

```
gardenforsalereport: Tgardenforsalereport;
```

```
implementation
```

```
uses unit45;
```

```
{ $R *.dfm }
```

```
procedure Tgardenforsalereport.FormCreate(Sender: TObject);
```



```

begin

end;

end.

unit Unit30;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  Tbuildingforsalereport = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    QRDBText1: TQRDBText;
    DetailBand1: TQRBand;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    QRLabel6: TQRLabel;
    QRLabel7: TQRLabel;
    QRLabel8: TQRLabel;
    QRLabel9: TQRLabel;
    QRLabel10: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRDBText4: TQRDBText;
    QRDBText5: TQRDBText;
    QRLabel11: TQRLabel;
    QRLabel12: TQRLabel;
    QRDBText6: TQRDBText;
    QRDBText7: TQRDBText;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  buildingforsalereport: Tbuildingforsalereport;

implementation

```

```

uses unit45;

{$R *.dfm}

procedure Tbuildingforsalereport.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit31;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  Tfieldforsalereport = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    QRLabel6: TQRLabel;
    QRLabel7: TQRLabel;
    QRLabel8: TQRLabel;
    QRLabel9: TQRLabel;
    QRLabel10: TQRLabel;
    QRLabel11: TQRLabel;
    QRLabel12: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRDBText4: TQRDBText;
    QRDBText5: TQRDBText;
    QRDBText6: TQRDBText;
    QRDBText7: TQRDBText;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

```

```

var
  fieldforsalereport: Tfieldforsalereport;

implementation

  uses unit45;

  {$R *.dfm}

  procedure Tfieldforsalereport.FormCreate(Sender: TObject);
  begin

  end;

end.

unit Unit32;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  Tvillaforsalereport = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    QRLabel6: TQRLabel;
    QRLabel7: TQRLabel;
    QRLabel8: TQRLabel;
    QRLabel9: TQRLabel;
    QRLabel10: TQRLabel;
    QRLabel11: TQRLabel;
    QRLabel12: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRDBText4: TQRDBText;
    QRDBText5: TQRDBText;
    QRDBText6: TQRDBText;
    QRDBText7: TQRDBText;
    procedure FormCreate(Sender: TObject);
  private

```

```

    { Private declarations }
public
    { Public declarations }
end;

var
    villaforsalereport: Tvillaforsalereport;

implementation

uses unit45;

{$R *.dfm}

procedure Tvillaforsalereport.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit33;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
    Tfarmforsalereport = class(TForm)
        QuickRep1: TQuickRep;
        PageHeaderBand1: TQRBand;
        DetailBand1: TQRBand;
        QRDBText1: TQRDBText;
        QRLabel1: TQRLabel;
        QRLabel2: TQRLabel;
        QRLabel3: TQRLabel;
        QRLabel4: TQRLabel;
        QRLabel5: TQRLabel;
        QRLabel6: TQRLabel;
        QRLabel7: TQRLabel;
        QRLabel8: TQRLabel;
        QRLabel9: TQRLabel;
        QRLabel10: TQRLabel;
        QRLabel11: TQRLabel;
        QRLabel12: TQRLabel;
        QRDBText2: TQRDBText;
        QRDBText3: TQRDBText;
        QRDBText4: TQRDBText;

```



```

QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  farmforsalereport: Tfarmforsalereport;

implementation

uses unit45;

{$R *.dfm}

procedure Tfarmforsalereport.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit34;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm34 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    QRLabel6: TQRLabel;
    QRLabel7: TQRLabel;

```

```

QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form34: TForm34;

implementation

  uses unit45;

{$R *.dfm}

procedure TForm34.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit35;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm35 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    QRLabel6: TQRLabel;

```

```

    QRLabel7: TQRLabel;
    QRDBText4: TQRDBText;
    QRDBText5: TQRDBText;
    QRDBText6: TQRDBText;
    QRDBText7: TQRDBText;
    procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form35: TForm35;

implementation

    uses unit45;

{$R *.dfm}

    procedure TForm35.FormCreate(Sender: TObject);
    begin

    end;

end.

unit Unit36;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
    TForm36 = class(TForm)
        QuickRep1: TQuickRep;
        PageHeaderBand1: TQRBand;
        ColumnHeaderBand1: TQRBand;
        DetailBand1: TQRBand;
        QRDBText1: TQRDBText;
        QRLabel1: TQRLabel;
        QRLabel2: TQRLabel;
        QRLabel3: TQRLabel;
        QRDBText2: TQRDBText;
        QRDBText3: TQRDBText;
        QRLabel4: TQRLabel;
        QRLabel5: TQRLabel;

```

```

QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form36: TForm36;

implementation

  uses unit45;

{$R *.dfm}

procedure TForm36.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit37;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm37 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRLabel4: TQRLabel;

```



```

QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form37: TForm37;

implementation

uses unit45;

{$R *.dfm}

procedure TForm37.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit38;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm38 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;

```

```

QRLabel4: TQRLabel;
QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form38: TForm38;

implementation

  uses unit45;

  {$R *.dfm}

  procedure TForm38.FormCreate(Sender: TObject);
  begin

  end;

end.

unit Unit39;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm39 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRDBText2: TQRDBText;

```

```

QRDBText3: TQRDBText;
QRLabel4: TQRLabel;
QRLabel5: TQRLabel;
QRLabel7: TQRLabel;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form39: TForm39;

implementation

  uses unit45;

  {$R *.dfm}

  procedure TForm39.FormCreate(Sender: TObject);
  begin

  end;

end.

unit Unit40;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm40 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;

```

```

QRLabel4: TQRLabel;
QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form40: TForm40;

implementation

  uses unit45;

  {$R *.dfm}

  procedure TForm40.FormCreate(Sender: TObject);
  begin

  end;

end.

unit Unit41;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm41 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;
    QRLabel4: TQRLabel;

```



```

QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRDBText2: TQRDBText;
QRDBText3: TQRDBText;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form41: TForm41;

implementation

  uses unit45;

  {$R *.dfm}

  procedure TForm41.FormCreate(Sender: TObject);
  begin

  end;

end.

unit Unit42;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm42 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRLabel3: TQRLabel;

```

```

QRDBText2: TQRDBText;
QRDBText3: TQRDBText;
QRLabel4: TQRLabel;
QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form42: TForm42;

implementation

  uses unit45;

  {$R *.dfm}

  procedure TForm42.FormCreate(Sender: TObject);
  begin

  end;

end.

unit Unit43;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

type
  TForm43 = class(TForm)
    QuickRep1: TQuickRep;
    PageHeaderBand1: TQRBand;
    ColumnHeaderBand1: TQRBand;
    DetailBand1: TQRBand;
    QRDBText1: TQRDBText;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;

```

```

QRLabel3: TQRLabel;
QRDBText2: TQRDBText;
QRDBText3: TQRDBText;
QRLabel4: TQRLabel;
QRLabel5: TQRLabel;
QRLabel6: TQRLabel;
QRLabel7: TQRLabel;
QRDBText4: TQRDBText;
QRDBText5: TQRDBText;
QRDBText6: TQRDBText;
QRDBText7: TQRDBText;
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form43: TForm43;

implementation

  uses unit45;

{$R *.dfm}

procedure TForm43.FormCreate(Sender: TObject);
begin

end;

end.

unit Unit44;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, Mask, DBCtrls;

type
  Tinformations = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;

```

```

Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Button1: TButton;
DBEdit1: TDBEdit;
DBEdit2: TDBEdit;
DBEdit3: TDBEdit;
DBEdit4: TDBEdit;
DBEdit5: TDBEdit;
DBEdit6: TDBEdit;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
DBEdit9: TDBEdit;
DBEdit10: TDBEdit;
DBEdit11: TDBEdit;
Bevel1: TBevel;
procedure Button1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  informations: Tinformations;

implementation

uses unit45;

{$R *.dfm}

procedure Tinformations.Button1Click(Sender: TObject);
begin
  dm.Table11.Edit;
  dm.Table11.Post;
end;

procedure Tinformations.FormCreate(Sender: TObject);
begin
  informations.ClientHeight:=573;
  informations.ClientWidth:=439;
end;

end.
unit Unit45;

interface

```



uses

SysUtils, Classes, DB, DBTables;

type

```
Tdm = class(TDataModule)
  dkhouse: TDataSource;
  dkshop: TDataSource;
  dsbuilding: TDataSource;
  dsvilla: TDataSource;
  dshouse: TDataSource;
  dsshop: TDataSource;
  dsfield: TDataSource;
  ds garden: TDataSource;
  dsplot: TDataSource;
  dsfarm: TDataSource;
  DataSource11: TDataSource;
  tkhouse: TTable;
  tkshop: TTable;
  tsbuilding: TTable;
  tsvilla: TTable;
  tshouse: TTable;
  tsshop: TTable;
  tsfield: TTable;
  ts garden: TTable;
  tsplot: TTable;
  tsfarm: TTable;
  Table11: TTable;
  procedure DataModuleCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

var

dm: Tdm;

implementation

{ \$R \*.dfm }

```
procedure Tdm.DataModuleCreate(Sender: TObject);
begin
```

```
end;
```

```
end.
```

unit Unit46;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, ExtCtrls, StdCtrls, Mask, DBCtrls, DB, DBTables;

type

```
TForm46 = class(TForm)
  Button1: TButton;
  Label1: TLabel;
  Label2: TLabel;
  Bevel1: TBevel;
  Bevel2: TBevel;
  Button2: TButton;
  DataSource1: TDataSource;
  Query1: TQuery;
  Edit1: TEdit;
  Edit2: TEdit;
  procedure Button1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure Button2Click(Sender: TObject);
  procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  procedure Edit2KeyPress(Sender: TObject; var Key: Char);
  procedure FormActivate(Sender: TObject);
  procedure Edit1Enter(Sender: TObject);
  procedure Edit2Enter(Sender: TObject);
  procedure Edit2Exit(Sender: TObject);
  procedure Edit1Exit(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

var

Form46: TForm46;

implementation

uses Unit1;

{ \$R \*.dfm }

```
function find(a:string;b:string):boolean;
begin
  find:=false;
  form46.query1.first;
  while not form46.query1.eof do
    if (a=form46.query1.fields[0].asstring)and(b=form46.query1.fields[1].asstring) then
      begin
        find:=true;
```

```
exit;  
end  
else  
form46.query1.next;  
end;
```

```
procedure TForm46.Button1Click(Sender: TObject);  
begin  
if find(edit1.text,edit2.Text) then  
begin  
form1.Show;  
form46.Visible:=false;  
end  
else  
application.MessageBox('please insert true UserName and Password','Warning',16);  
end;
```

```
procedure TForm46.FormCreate(Sender: TObject);  
begin  
borderIcons:=borderIcons-[bisystemmenu,bimaximize,biminimize];  
Form46.ClientHeight:=282;  
Form46.ClientWidth:=355;  
end;
```

```
procedure TForm46.Button2Click(Sender: TObject);  
begin  
form46.Close;  
end;
```

```
procedure TForm46.Edit1KeyPress(Sender: TObject; var Key: Char);  
begin  
if(key=#13)then Edit2.SetFocus;  
end;
```

```
procedure TForm46.Edit2KeyPress(Sender: TObject; var Key: Char);  
begin  
if(key=#13)then Button1.SetFocus;  
end;
```

```
procedure TForm46.FormActivate(Sender: TObject);  
begin  
edit1.SetFocus;  
end;
```

```
procedure TForm46.Edit1Enter(Sender: TObject);  
begin  
if sender is tedit then tedit(sender).Color:=clMoneyGreen;  
end;
```

```
procedure TForm46.Edit2Enter(Sender: TObject);  
begin
```

```
if sender is tedit then tedit(sender).Color:=clMoneyGreen;
end;
```

```
procedure TForm46.Edit2Exit(Sender: TObject);
begin
  if sender is tedit then tedit(sender).Color:=clMenuBar;
end;
```

```
procedure TForm46.Edit1Exit(Sender: TObject);
begin
  if sender is tedit then tedit(sender).Color:=clMenuBar;
end;
```

```
end.
```

```
unit Unit47;
```

```
interface
```

```
uses
```

```
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, Buttons, ExtCtrls, Grids, DBGrids, DB,
  DBTables, jpeg;
```

```
type
```

```
TForm47 = class(TForm)
  Label1: TLabel;
  Label2: TLabel;
  DataSource1: TDataSource;
  Query1: TQuery;
  DBGrid1: TDBGrid;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  BitBtn3: TBitBtn;
  BitBtn4: TBitBtn;
  DBEdit1: TDBEdit;
  DBEdit2: TDBEdit;
  BitBtn5: TBitBtn;
  BitBtn6: TBitBtn;
  Bevel1: TBevel;
  procedure BitBtn1Click(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);
  procedure BitBtn3Click(Sender: TObject);
  procedure BitBtn4Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure BitBtn5Click(Sender: TObject);
  procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
  procedure BitBtn6Click(Sender: TObject);
private
  { Private declarations }
```



```

public
  { Public declarations }
end;

var
  Form47: TForm47;

implementation

uses unit46;

{$R *.dfm}

procedure TForm47.BitBtn1Click(Sender: TObject);
begin
  Query1.Edit;
  DBEdit1.SetFocus;
end;

procedure TForm47.BitBtn2Click(Sender: TObject);
begin
  Query1.Post;
end;

procedure TForm47.BitBtn3Click(Sender: TObject);
var
  a:word;
begin
  a:=Application.MessageBox('Are you sure?','Warning',36);
  if(a=IDYES)then
  begin
    Query1.Delete;
  end;
end;

procedure TForm47.BitBtn4Click(Sender: TObject);
begin
  Query1.Cancel;
  form47.Close;
end;

procedure TForm47.FormCreate(Sender: TObject);
begin
  borderIcons:=borderIcons-[bisystemmenu,bimaximize,biminimize];
  DBEdit1.Text:="";
  dbedit2.Text:="";
  Form47.ClientHeight:=278;
  Form47.ClientWidth:=365;
end;

```

```

procedure TForm47.BitBtn5Click(Sender: TObject);
begin
  DBEdit1.Text:="";
  dbedit2.Text:="";
  DBEdit1.SetFocus;
  Query1.Insert;
end;

procedure TForm47.DBEdit1KeyPress(Sender: TObject; var Key: Char);
begin
  if(key=#13)then DBEdit2.SetFocus;
end;

procedure TForm47.BitBtn6Click(Sender: TObject);
begin
  DBEdit1.Clear;
  DBEdit2.Clear;
  Query1.Cancel;
end;

end.

```