

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Customer Registration Program by Using Delphi

Graduation Project

COM 400

Student: Turgay BALTEPE

Supervisor: Umit SOYER

Nicosia – 2008

ACKNOWLEDGEMENT

First of all, I would like give my special thanks to my supervisor Umit SOYER. 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.

TURGAY BALTEPE

ABSTRACT

The aim of this program is registering the customers of a hostel and when the residing ends, making a registry out for the customer, calculating the total price and displaying them as active or inactive. As a programming language Delphi was used and as a database Access was used.

I chose Delphi as programming language because Delphi speeds Win32 development by combining Delphi's proven visual Rapid Application Development approach for accelerated Win32 development with support for Windows Vista, AJAX, and streamlined database connectivity. In the real world, developers need to be able to develop applications that run on multiple platforms, not just the latest and greatest. Most new machines come with Windows Vista, while existing machines will continue running Windows 2000 or XP. Developers must support this mixed-use environment, because they can't count on their organization or customers upgrading en masse. They have to meet the demand for critical new technologies and trends in marketplace by including support for these technologies in their applications, but they want to retain the flexibility of developing on the platform that is most productive for them. Access database is one of the world's most popular database because of its consistent fast performance, high reliability and ease of use. In Microsoft Access 2003, you can view information on dependencies between database objects. Viewing a list of objects that use a specific object helps maintain a database over time and avoid errors related to missing record sources. For example, the Quarterly Orders query in the Sales database is no longer needed, but before deleting it, you might want to find out which other objects in the database use the query. Then, you could either change the record source of the dependent objects, or delete them, before deleting the Quarterly Orders query. Viewing a complete list of dependent objects helps you save time and minimize errors. In addition to viewing the list of objects that are bound to a selected object, you can also view the objects that are being used by the selected object. Macros, modules, and data access pages are not searched for dependencies. Access projects do not support this feature.

Table of Contents

ACKNOWLEDGEMENT.....	I
ABSTRACT.....	II
TABLE OF CONTENTS	III
INTRODUCTION	1

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.....	59
2.11.5	Adding the Report to Your Project.....	60

CHAPTER THREE : USER MANUAL

3.1	Enter Form.....	61
3.2	Main Form.....	62
3.3	HoneyHill Hostel Form.....	63
3.3.1	Customer Registration Page.....	64
3.3.2	Customer Registration Out Page.....	65
3.3.3	Search Page.....	66
3.3.4	Registries General Status.....	67
3.3.5	Rooms General Status.....	68

CONCLUSION.....	69
-----------------	----

REFERENCES.....	70
-----------------	----

APPENDIX.....	71
---------------	----

INTRODUCTION

Honeyhill Hostel Automation Software designed as simple as possible. There is no animations that drains CPU and reduces system performance. User can open another program without closing the program. It uses three tables in the database which are related each other.

The user basically register the customers who will stayed in the hostel and by making this program shows them as active when a necessary search is done. If a customer wants to leave, the user make them out and by making this program shows it inactive when a necessary search is done.

CHAPTER 1

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](http://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":

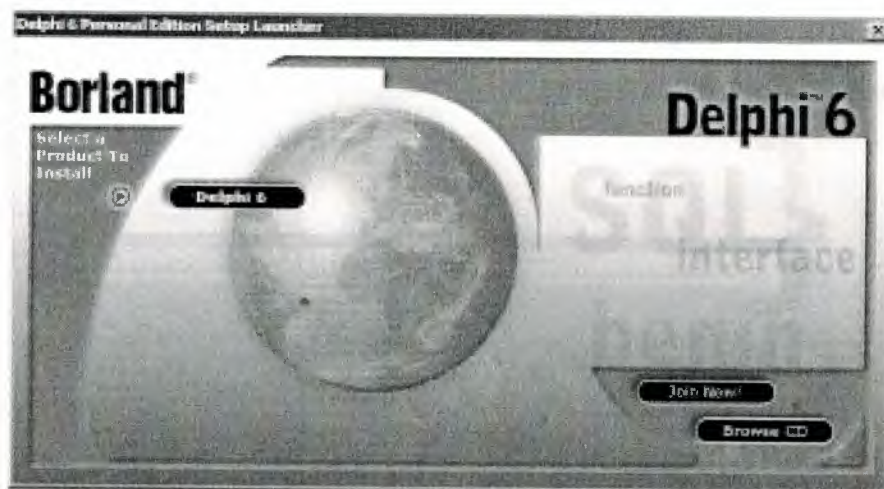


Figure 1.1 The Select Page For Start Installation

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 will popup:

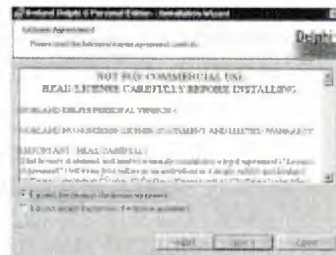


Figure 1.3 License Agreement Screen

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.

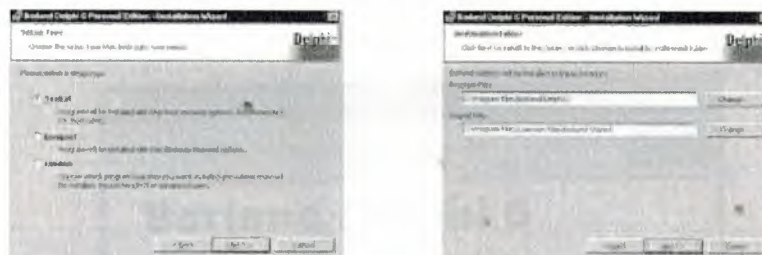


Figure 1.4 SetUp Type and Destination Folder Screen

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.

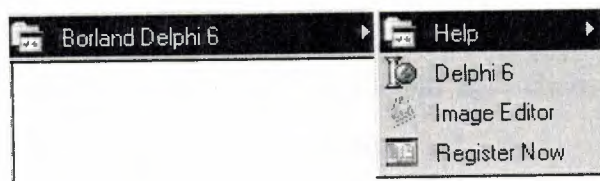


Figure 1.5 Start Menu

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)

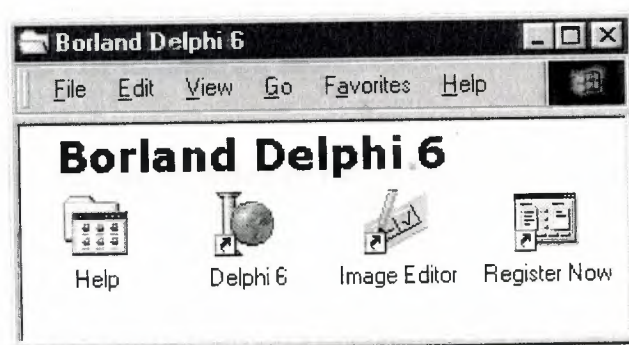


Figure 1.6 Borland Delphi 6 Folder

- 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.

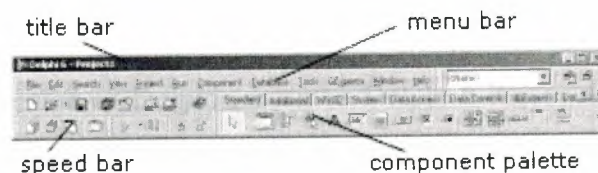


Figure 1.8 Menu, Title, Speed Bar & Component Palette

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.

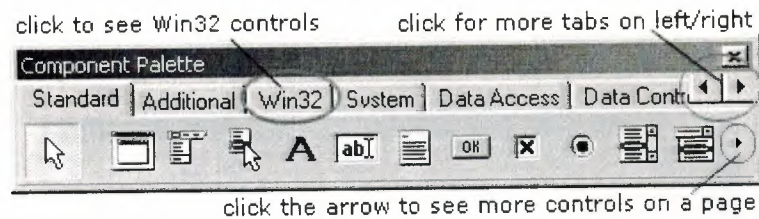


Figure 1.9 Component Palates

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

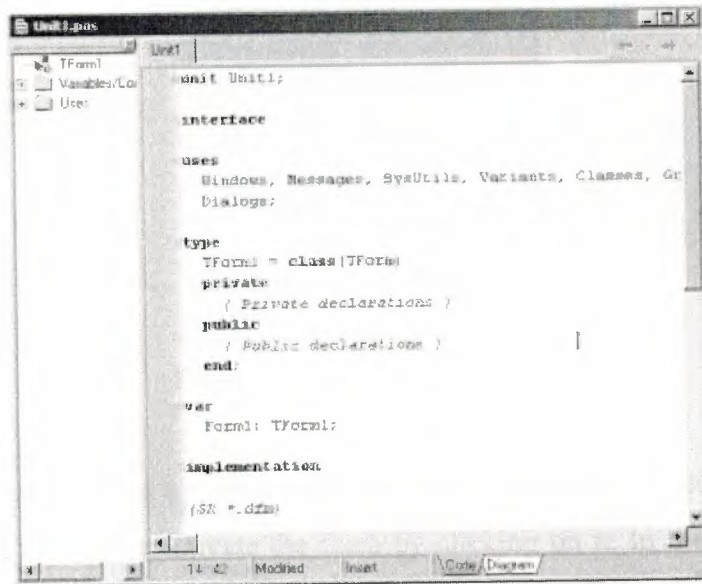


Figure 1.10 Code Editor Window

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.

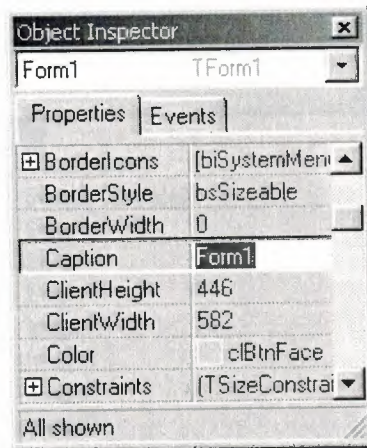


Figure 1.11 Object Inspector

For example, each form has a Caption (the text that appears on its 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.

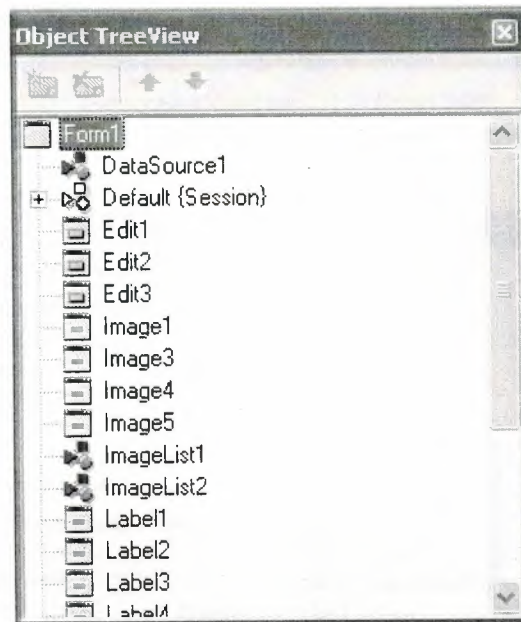


Figure 1.12 Object Tree View

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.

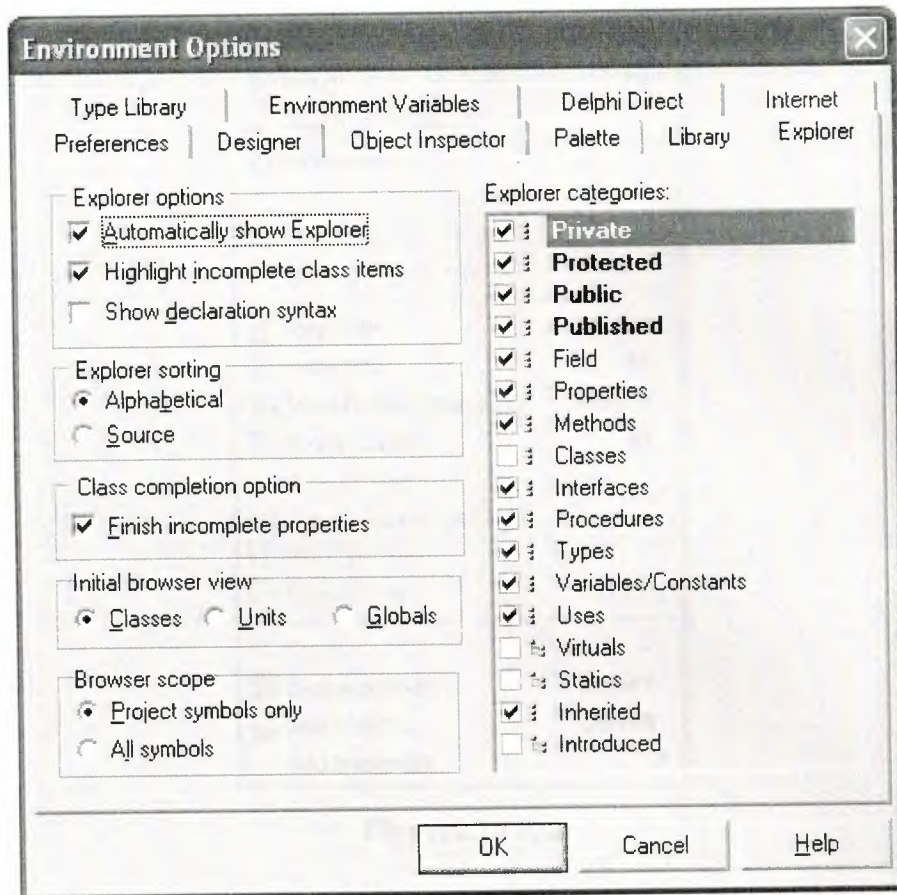


Fig.1.13 Class Completion

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.

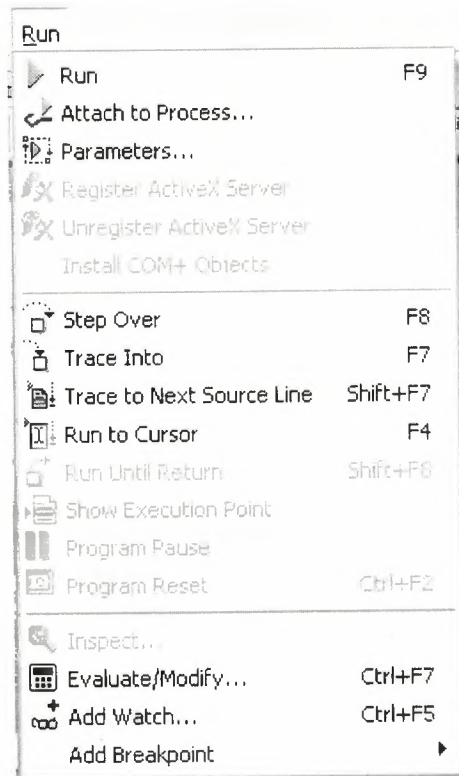


Figure1.14 Run

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.

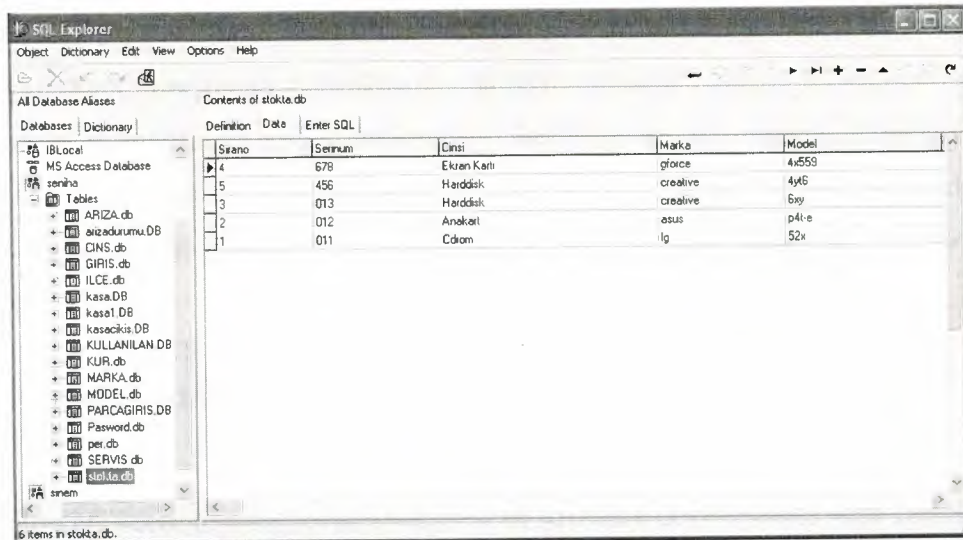


Figure 1.15 SQL Explorer

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.

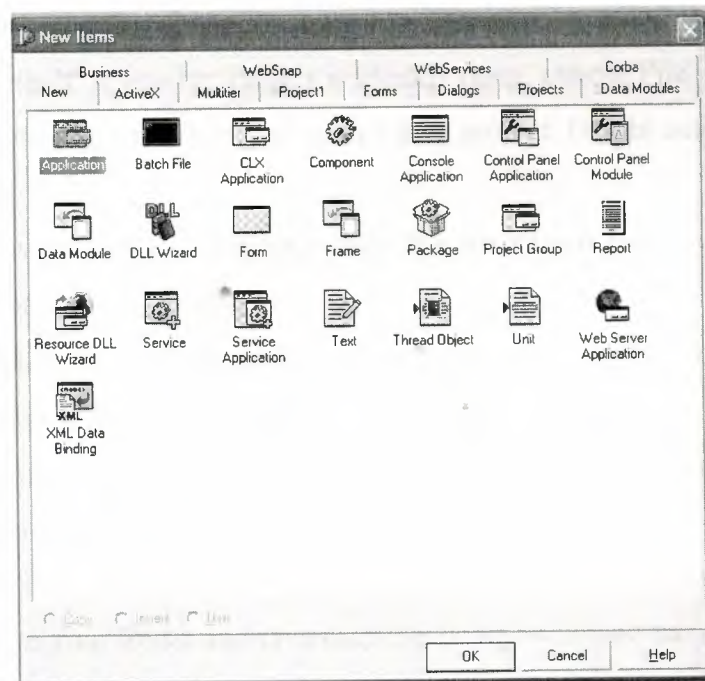


Figure 1.16 New Item

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.

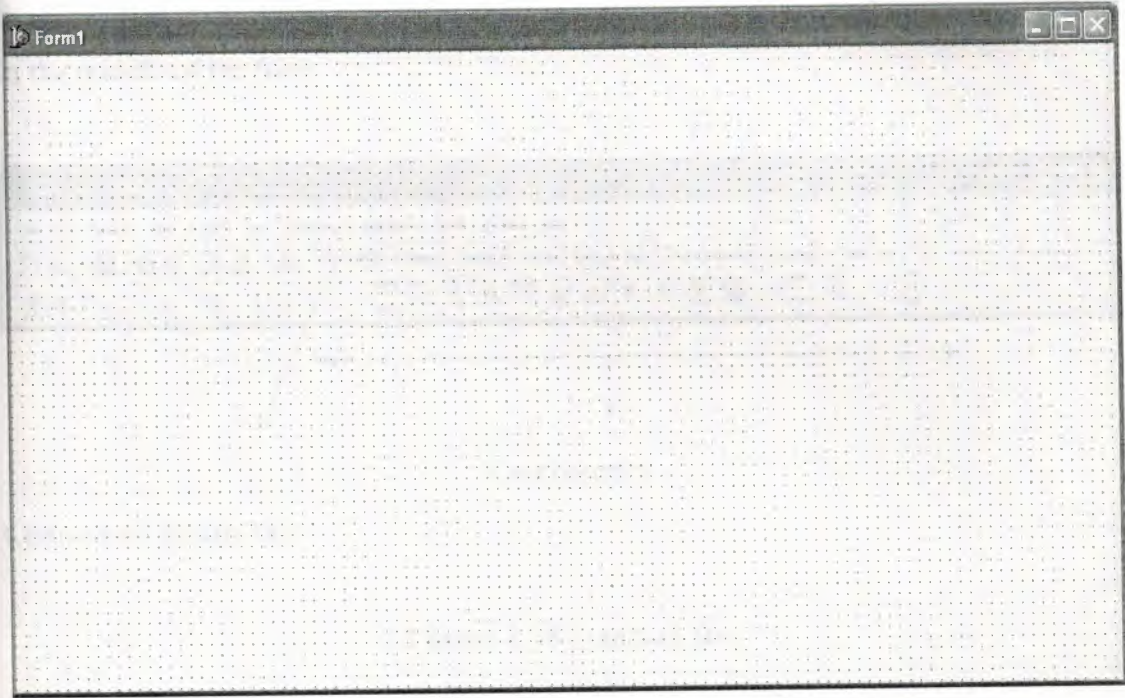


Figure 1.17 Form Screen

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

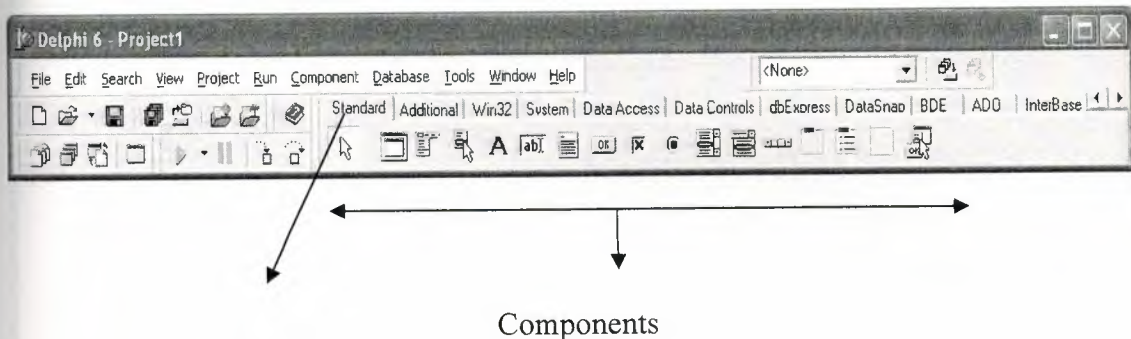
When you use the Object Inspector to set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called design-time settings.

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.



Component palette tabs

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.

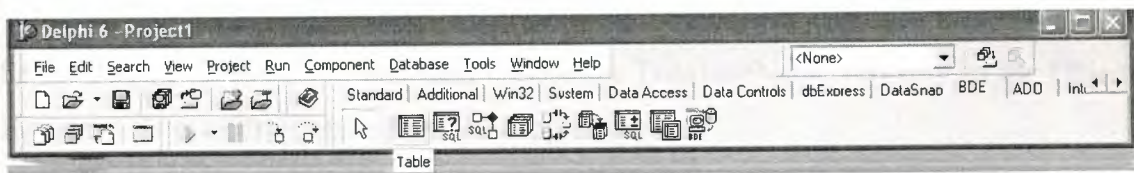


Figure 1.19 BDE Component palette

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.)

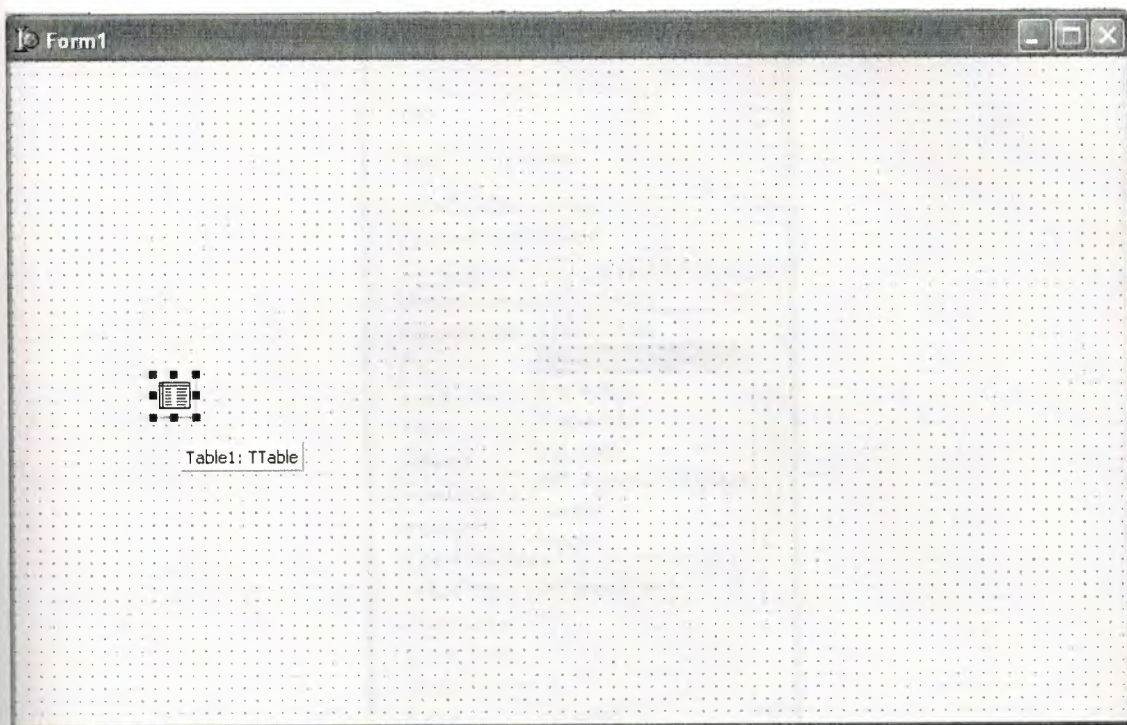


Figure 1.20 Table in the Form

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.

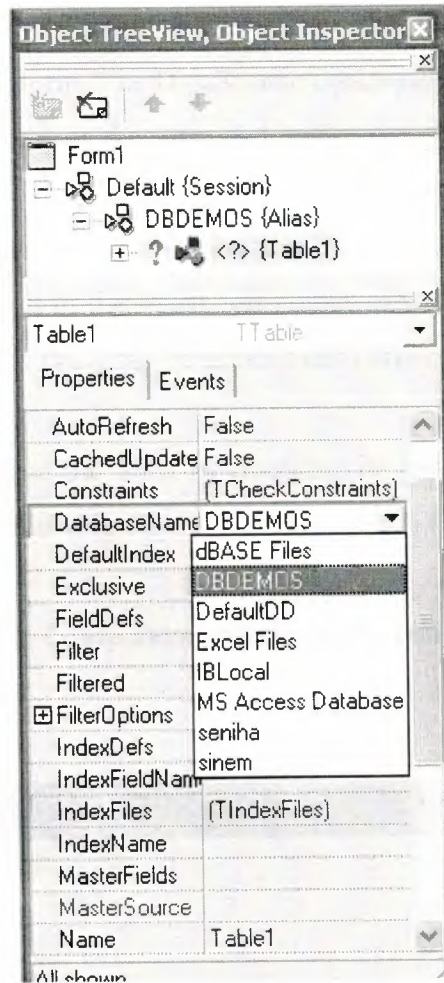


Figure 1.21 Select DatabaseName

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.

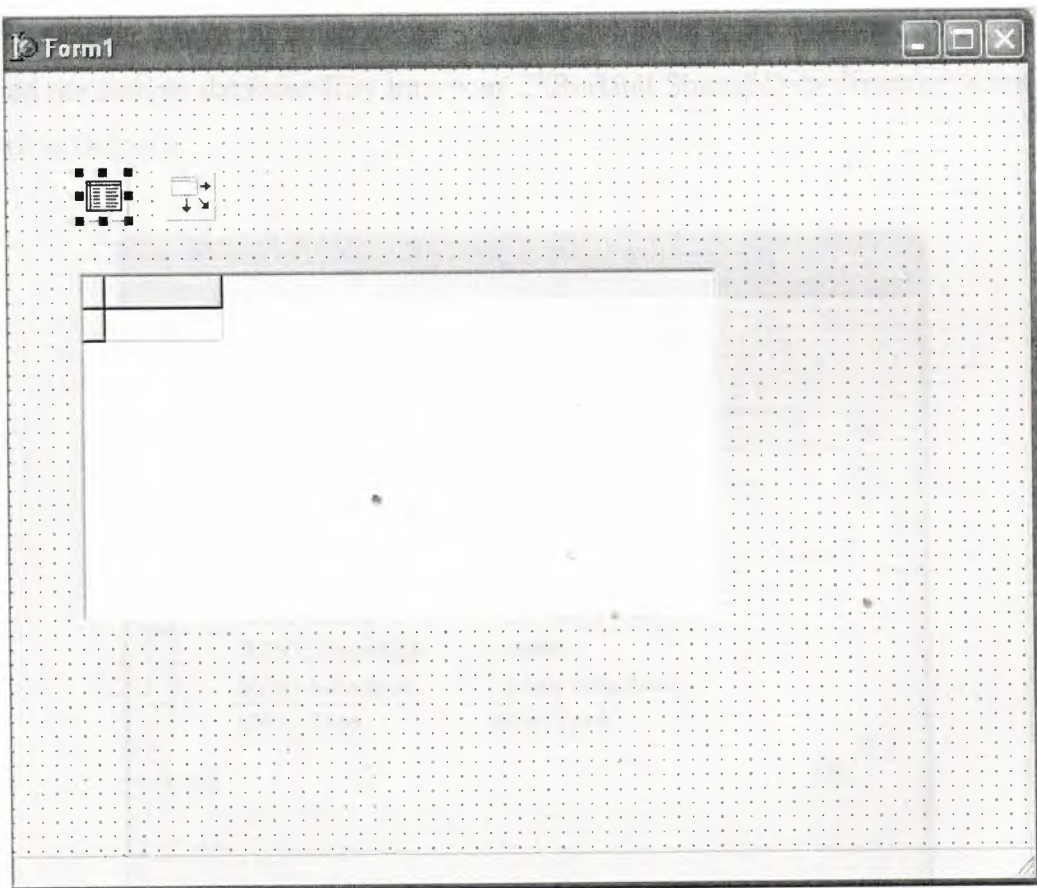


Figure 1.22 DBGrid in the Form

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.)

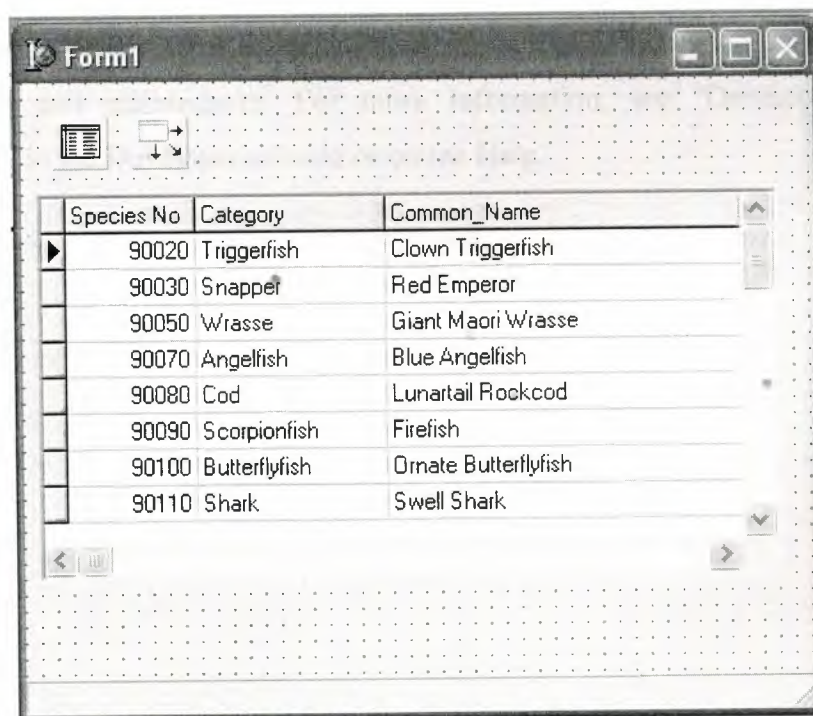


Figure 1.23 Show Table

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 2

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.

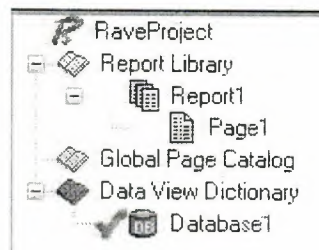


Figure 2.1 Project Tree

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.

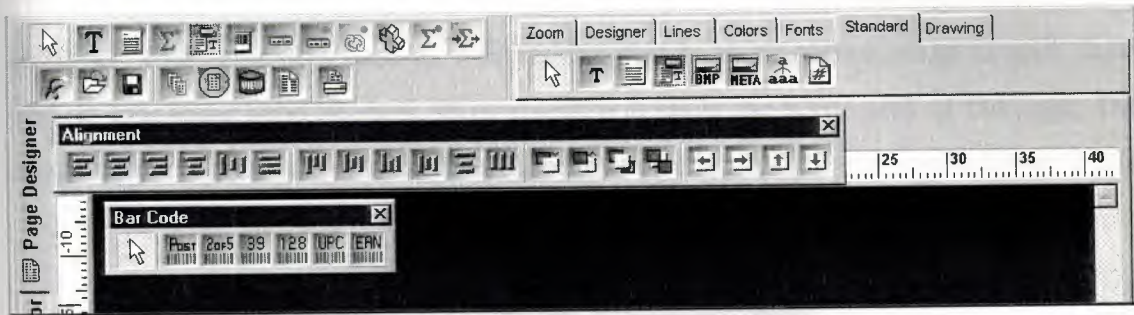


Figure 2.2 Toolbars

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.

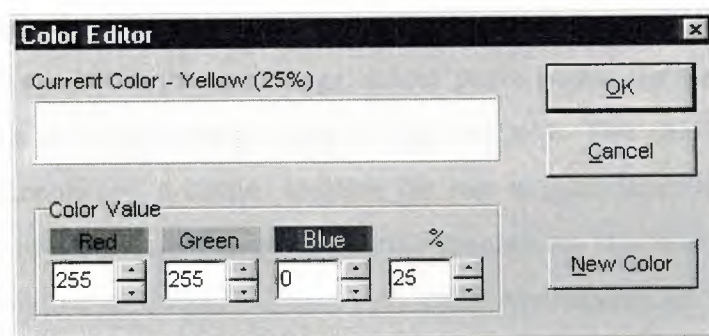


Figure 2.8 Colors Editor

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.

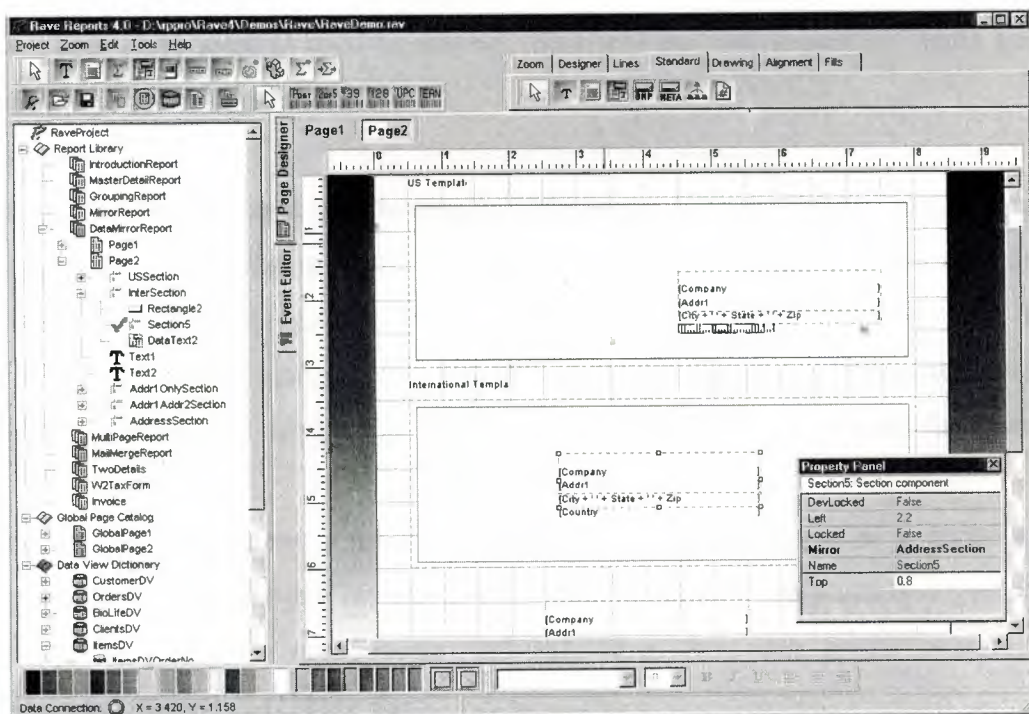


Figure 2.11 Mirror Report Example

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.

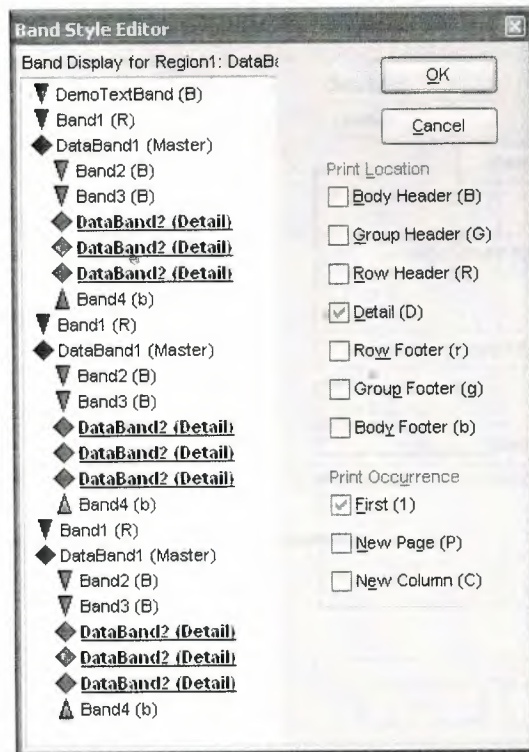


Figure 2.15 Band Style Editor

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.

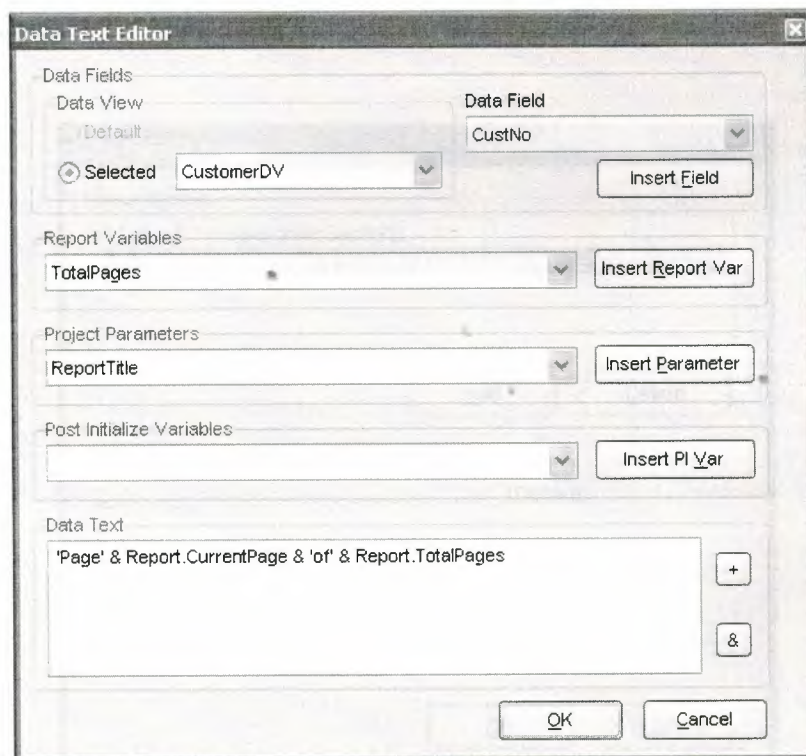


Figure 2.16 Data Text Editor

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.

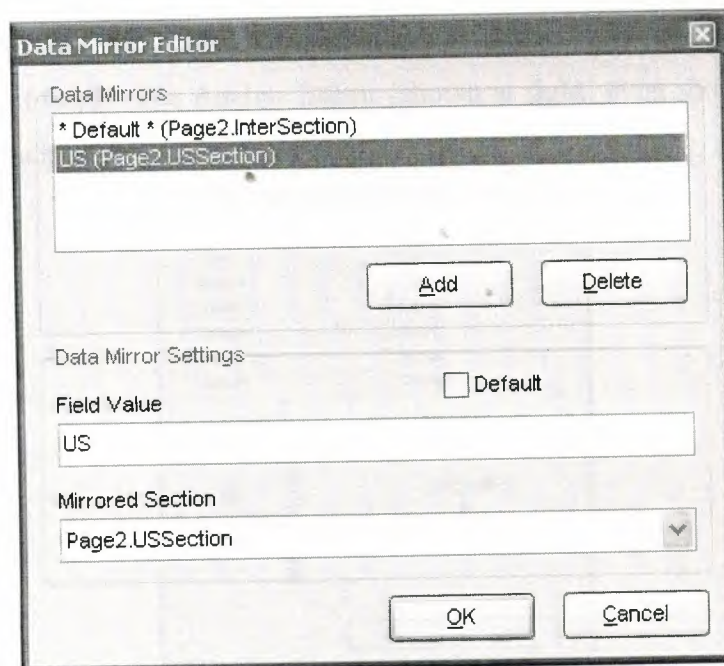


Figure 2.17 Data Mirror Editor

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.

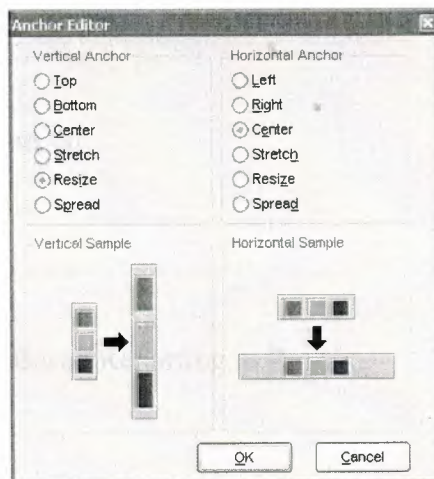


Figure 2.19 Anchor Editor

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:

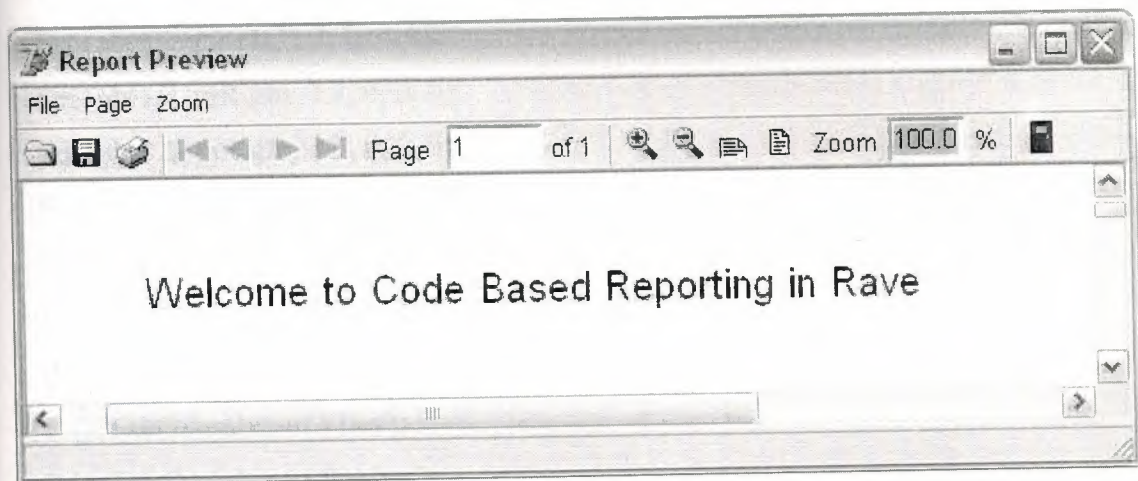


Figure 2.20 Report Preview

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);
            PrintTab(Format('%u',[NumFiles]));
          end
        end
      except
      end
    end
  end
end

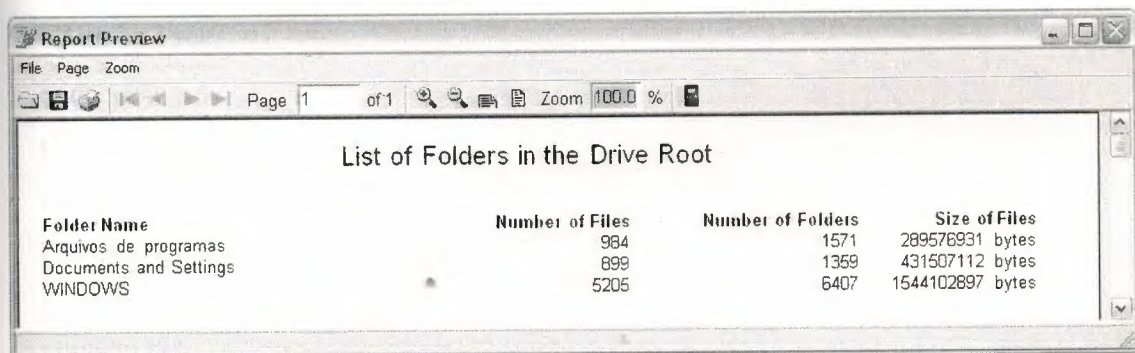
```

```

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, approximately, 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

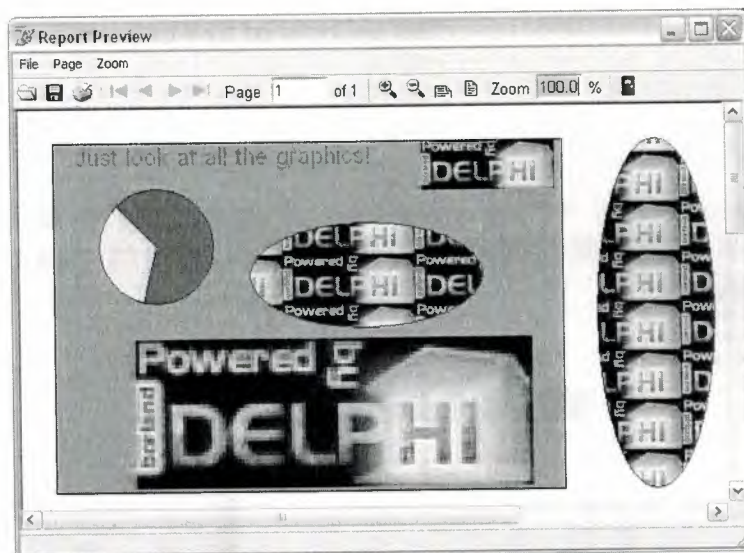


Figure 2.22 Report Preview

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:

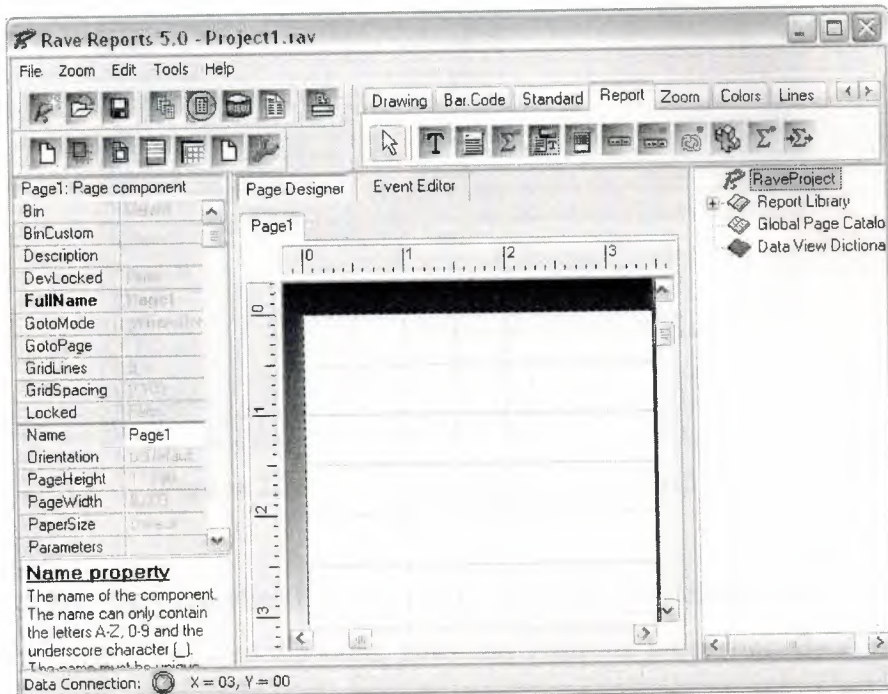


Figure 2.23 Rave Visual Designer

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 useful 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 its 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
Font	Arial,15
FontJustify	piLeft
FontMirror	
Left	1,000
Locked	False
Mirror	
Name	Text1
Rotation	0
Tag	0
Text	Welcome to Rave Reports Visual
Top	1,000
Truncate	True
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:

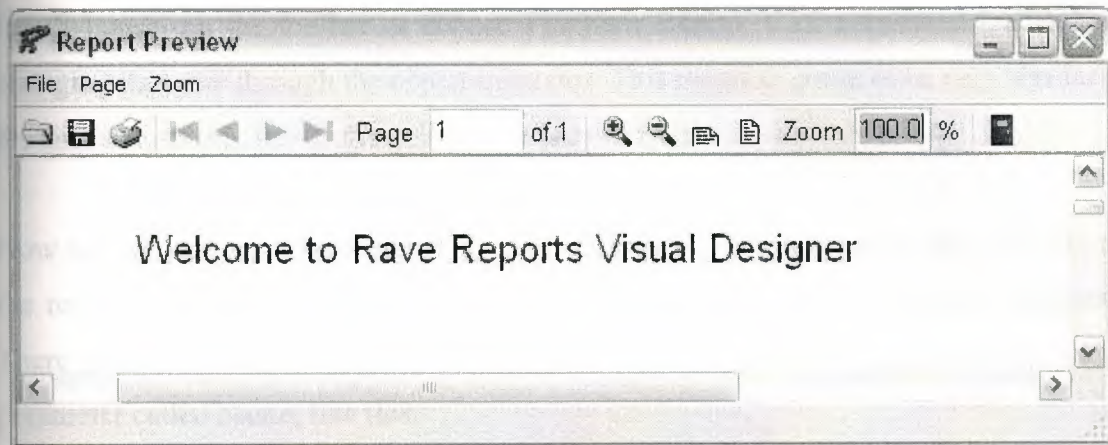


Figure 2.25 Report Preview

Tip: If you Close and Open your project before executing, you won't need to 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:

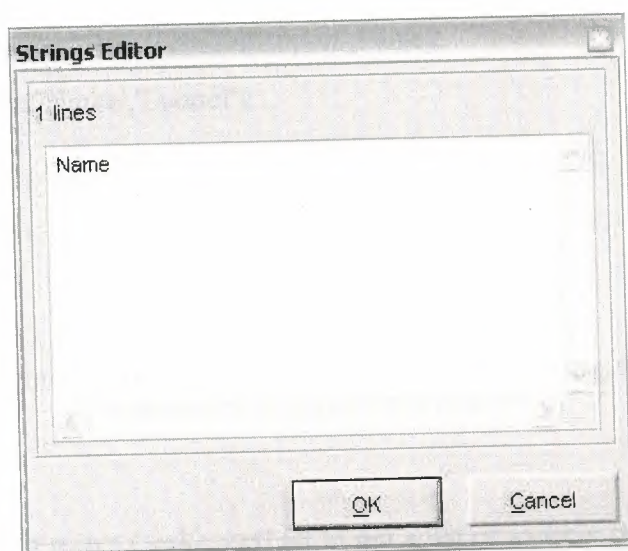


Figure 2.26 Strings Editor

Parameters can be printed using a DataText component, available in the Report tab of the component palette. 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:

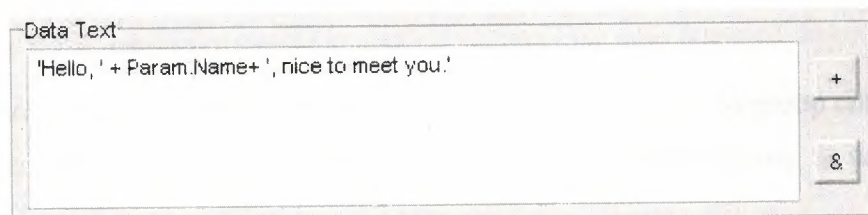


Figure 2.27 Data Text Sample

Here's the result:

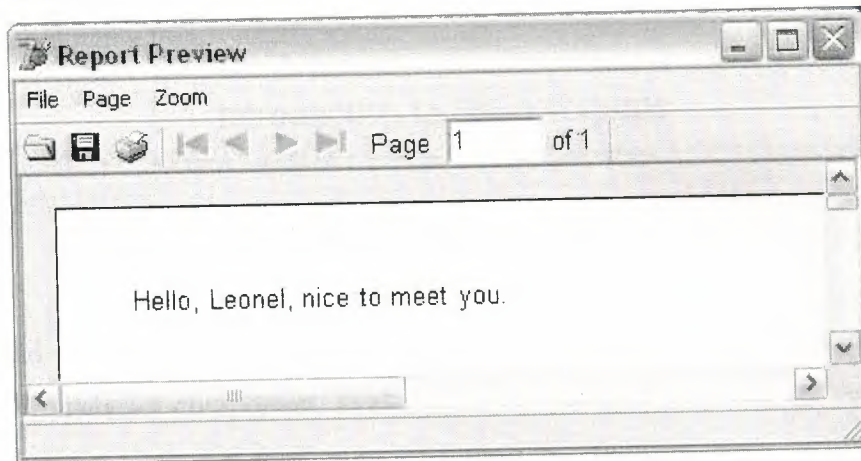


Figure 2.28 Report Preview

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:

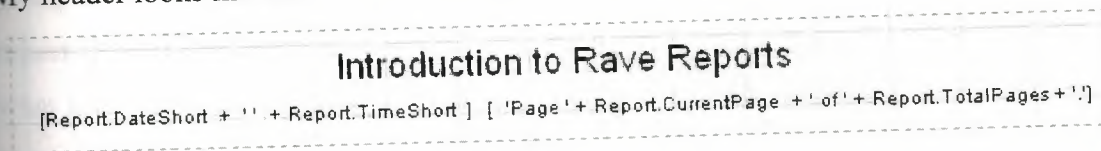


Figure 2.29 Header Sample

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:

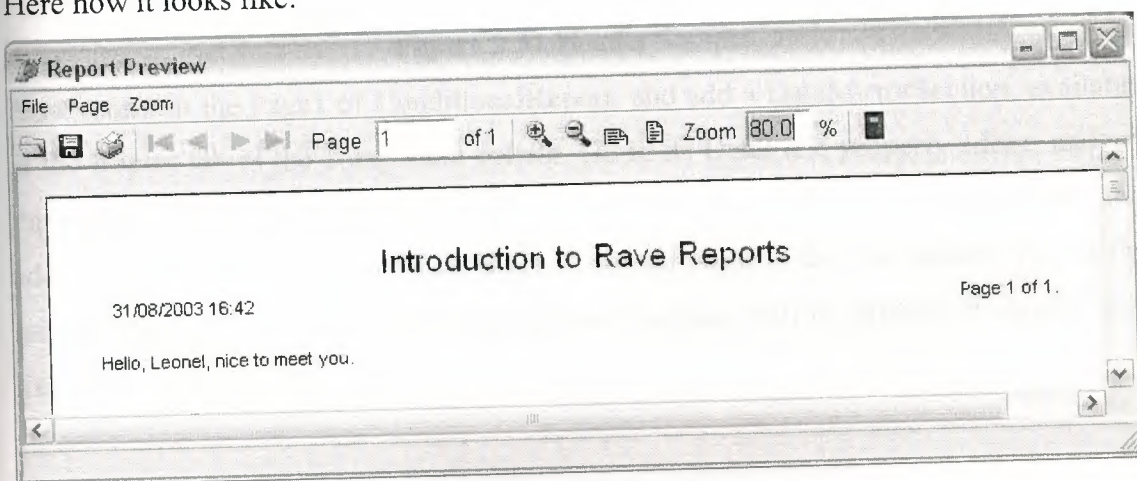


Figure 2.30 Report Preview

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:

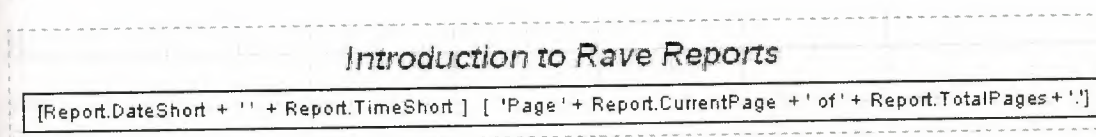


Figure 2.31 Header Sample

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:

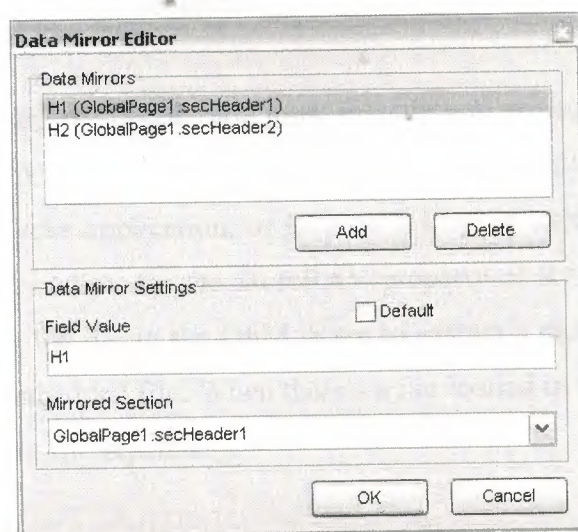


Figure 2.32 Data Mirror Editor

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:

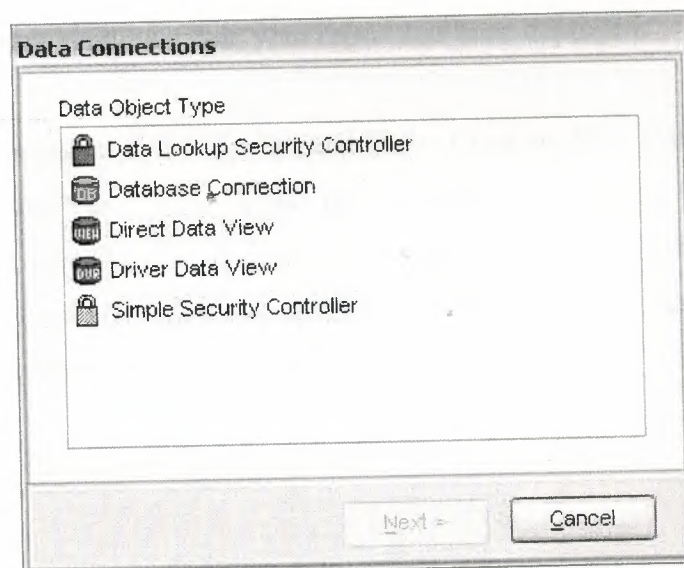


Figure 2.33 Data Connection Window

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:

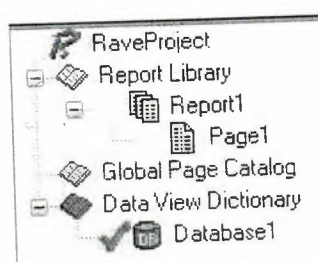


Figure 2.34 Project Tree View

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:

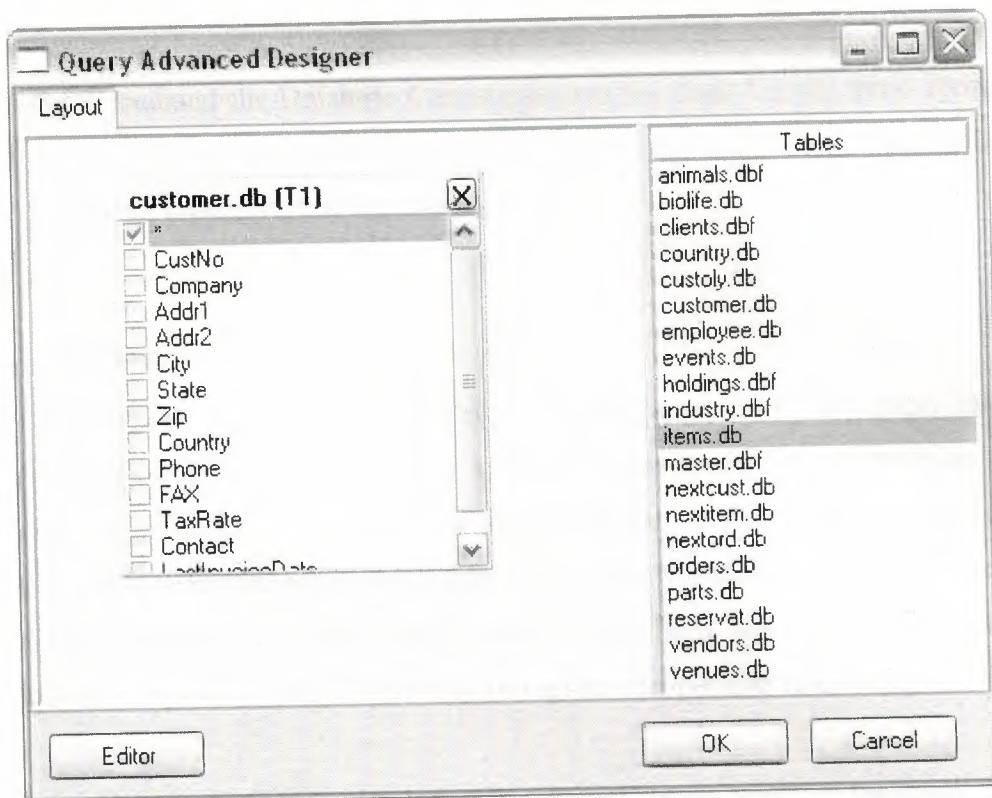


Figure 2.35 Query Advanced Designer Window

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:

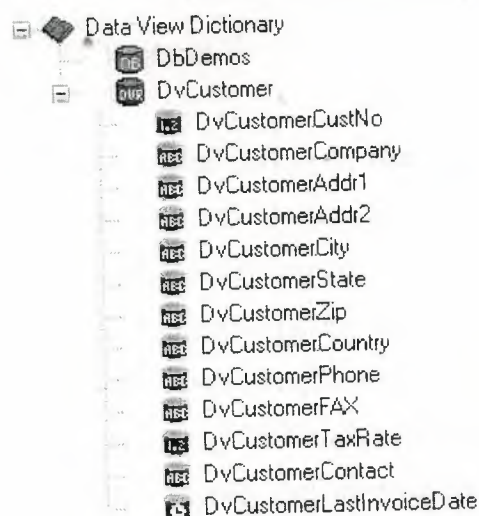


Figure 2.36 Project Tree View

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 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:

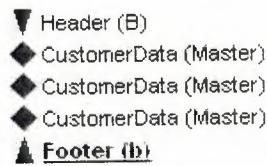


Figure 2.37 Band Display

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:

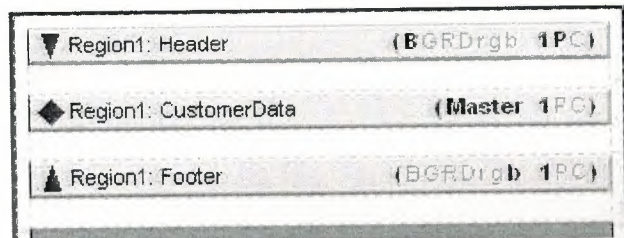


Figure 2.38 Editor Sample

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 3

3 USER MANUAL

3.1 Enter Form

When user executes this program, first password enter screen appears. In this screen user enters a password as seen in the figure 3.1 below. If the user wants to leave the program without entering , this can be done by clicking the EXIT button.



Figure 3.1

3.2 Main Form

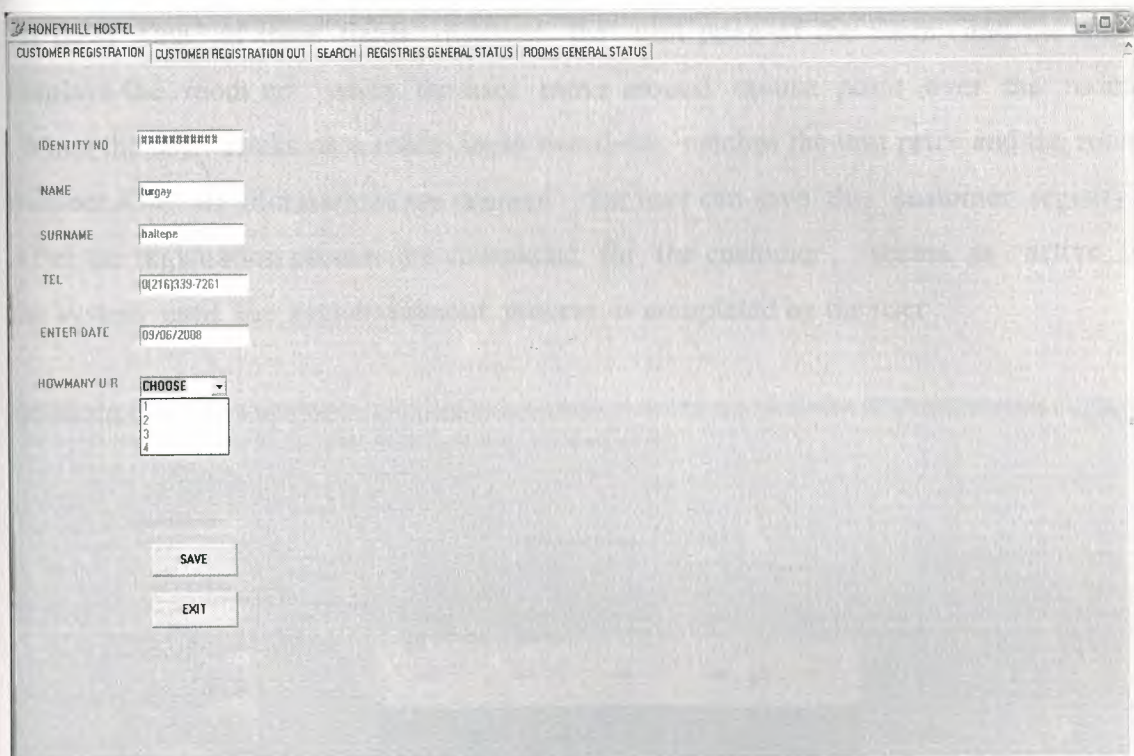
After password are entered, main screen appears as seen in the figure 3.2 below. RENT A ROOM and EXIT buttons exists in this form. When the user clicks the RENT A ROOM button, HONEYHILL HOSTEL form shows up as you may see in the figure 3.3 below



Figure 3.2

3.3 Honeyhill Hostel Form

There are five pages in this form .First page is the Customer Registration page as you may see in the figure 3.4 below.And the other pages follow this pages consecutively as Customer Registration Out,Search,Registries General Status,Rooms General Status.



The screenshot shows a web application window titled "HONEYHILL HOSTEL". The window has a navigation bar with five tabs: "CUSTOMER REGISTRATION" (active), "CUSTOMER REGISTRATION OUT", "SEARCH", "REGISTRIES GENERAL STATUS", and "ROOMS GENERAL STATUS". The main content area is a form for customer registration. It contains the following fields and controls:

- IDENTITY NO**: A text input field containing "XXXXXXXXXX".
- NAME**: A text input field containing "lurgay".
- SURNAME**: A text input field containing "haltepe".
- TEL**: A text input field containing "08216339-7261".
- ENTER DATE**: A text input field containing "03/06/2008".
- HOWMANY U R**: A dropdown menu with "CHOOSE" selected. Below it is a list box showing the numbers 1, 2, 3, and 4.
- SAVE**: A button located below the list box.
- EXIT**: A button located below the SAVE button.

Figure 3.3

3.3.1 Customer Registration Page

In this page, there are six empty fields which are used for entering customer information. One of these fields has fixed numbers as 1,2,3,4 related with the number of the people who will be stayed in a room. When user clicks on one of these numbers, a panel shows up on the right of the page with the rooms inside of it and the rooms are displayed as blue if the chosen number suits to room, because some of rooms maybe full, if it happens like that the rooms are displayed as white. The panel includes 2 empty fields and 20 rooms as rectangles with written room numbers on them. One of these fields displays the unit price of the room and the other displays the room no when the user move around mouse point over the rooms. When the user clicks on a room these two fields catches the unit price and the room number. After all informations are ensured, the user can save this customer registry. After the registration process are completed for the customer, seems as active in the system until the registration out process is completed by the user.

	1 FLOOR	2 FLOOR	3 FLOOR	4 FLOOR	5 FLOOR	
FOR 1 PERSON	o101	o201	o301	o401	o501	
FOR 2 PEOPLE	o102	o202	o302	o402	o502	
FOR 3 PEOPLE	o103	o203	o303	o403	o503	
FOR 4 PEOPLE	o104	o204	o304	o404	o504	

Figure 3.4

3.3.2 Customer Registration Out Page

This page was designed to make customer registries out or inactive. If the user enters into the field of name one of the first letters that customer names have, all customer infos as name, surname, roomno and customers.registerno shows up on the black screen. If a complete customer name is entered into the name field, only one line shows up on the screen which is related with this customer. After the line appears, the user clicks on the line and the empty fields which are on the left of the screen, catches the informations about the customer as surname, leaving date, unit price of the room, enter date, howmany people have stayed in the room, howmany days have been stayed on the hostel and the total price which is calculated by the program. When the user press the Save Registration Out button, the process of making customer inactive is being performed. If Registration Out Cancel button is pressed, the process of making customer inactive is canceled and the all fields are cleared.

name	surname	roomno	howmany	customers.registerno
metin	baltepe	304	4	57

Figure 3.5

3.3.3 Search Page

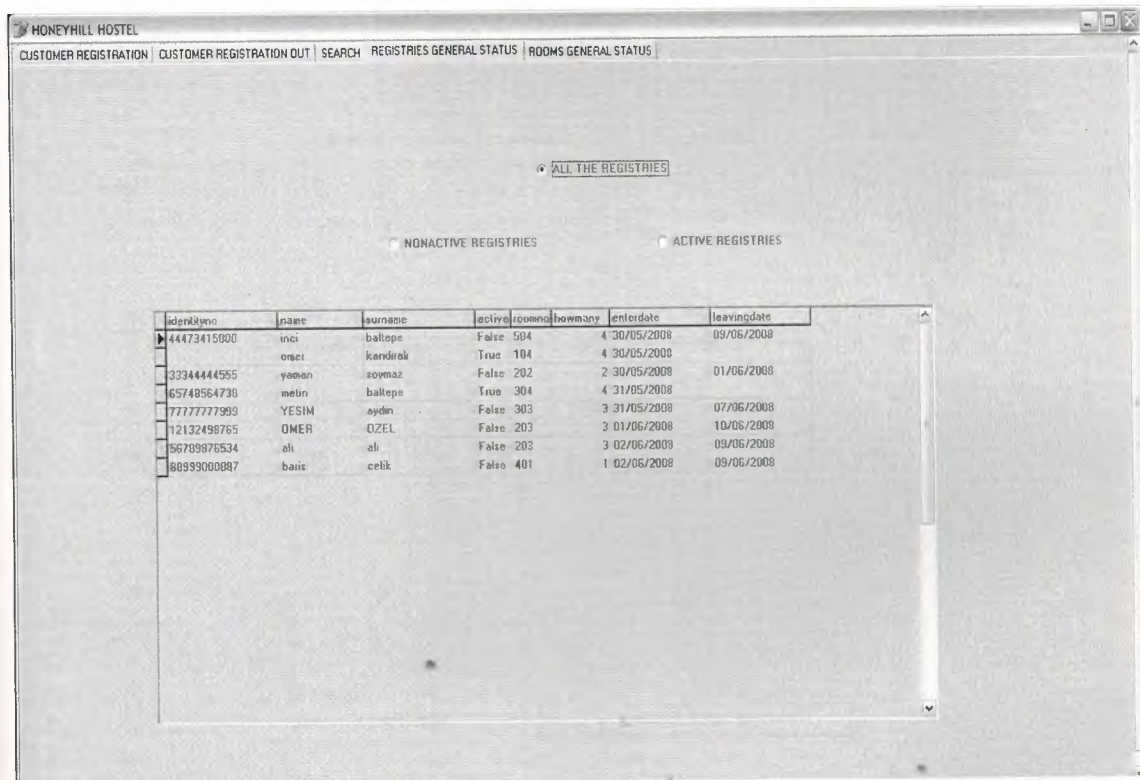
There are options in this page to search customers as search by roomno, search by name and search by the dates. It is possible to see the customers as active or inactive on the screen.

idenfno	name	surname	tel	active	roomno	howmany	enterdate	leavedate
---------	------	---------	-----	--------	--------	---------	-----------	-----------

Figure 3.6

3.3.4 Registries General Status

There are three options to see the registries. First the user can choose the all the registries box and the all registries appear on the screen. If the user wants to see the active registries, active registries box is choosed and the all active registries show up on the screen. The user can use the nonactive registries box which is the last option and after that user sees the all nonactive registries on the screen.



idenk/yno	name	surname	active	roomno	howmany	enterdate	leavingdate
44473415000	inci	baltepe	False	504	4	30/05/2008	09/06/2008
	onaci	kardirak	True	104	4	30/05/2008	
33344444555	yaman	soymaz	False	202	2	30/05/2008	01/06/2008
65748564730	metin	baltepe	True	304	4	31/05/2008	
77777777999	YESIM	aydin	False	303	3	31/05/2008	07/06/2008
12132498765	OMER	OZEL	False	203	3	01/06/2008	10/06/2008
56789876534	ali	ali	False	203	3	02/06/2008	09/06/2008
8899000887	baris	celik	False	401	1	02/06/2008	09/06/2008

Figure 3.7

3.3.5 Rooms General Status

In page you can see the rooms fullness status with the ratios. There are 20 panels on the page with shape of rectangles. If the user presses the 'Show the fullness ratio button', the ratios appear in the rectangles. If the roomno 102 is full, the ratio appears as 2/2 or if the user presses the other button which is 'Show the being empty ratio' and if the room 102 is full, it appears as 0/2. If the room which is for 4 people has 3 people inside of it the fullness ratio appears for this room as 3/4 and the being empty ratio appears as 1/4.

1ST FLOOR	2ND FLOOR	3RD FLOOR	4TH FLOOR	5TH FLOOR
101 1/1	201 1/1	301 1/1	401 1/1	501 1/1
102 2/2	202 2/2	302 2/2	402 2/2	502 2/2
103 3/3	203 3/3	303 3/3	403 3/3	503 3/3
104 0/4	204 1/4	304 0/4	404 0/4	504 4/4

SHOW THE FULLNESS RATIO SHOW THE BEING EMPTY RATIO

Figure 3.8

CONCLUSION

It was a really good decision for me to use Delphi as programming language and Access as a database. Because I succeeded everything that I have planned before started to my project. In the most situations I used the internet and searched some books. I had some difficulties with the SQL commands and programming in this project. But with the help of the some books and internet, I accomplished.

Because of I am not advanced in delphi programming, it was difficult to get control in the programming section. But this project helped me to develop myself with the delphi.

This program is enough for a hostel to register its customers and control this registration. With making some extra additions, this program can be made more functional.

I want to learn another language to jump level in the programming as soon as possible.

REFERENCES

<http://www.w3schools.com/sql>

<http://www.delphibasics.co.uk>

Delphi 7 Ezel Balkan Book

Delphi 7 Zirvedeki Beyinler Nihat Demirli Yuksel Inan (ebook)

Borland Delphi 6 for Windows (e Book)

Mastering Delphi 6 – Mastering Delphi 7

APPENDIX

Program Codes

unit turgay;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ADODB, DB, Grids, DBGrids, Mask, ExtCtrls, ComCtrls,
TabNotBk, DBCtrls;

type

TForm1 = class(TForm)

ads1: TADODataset;

adq1: TADOQuery;

ds1: TDataSource;

TabbedNotebook1: TTabbedNotebook;

Label4: TLabel;

Label8: TLabel;

Label9: TLabel;

Label10: TLabel;

Label13: TLabel;

Label14: TLabel;

Edit5: TEdit;

Edit7: TEdit;

MaskEdit3: TMaskEdit;

MaskEdit4: TMaskEdit;

Button4: TButton;

ComboBox1: TComboBox;

Label1: TLabel;

Edit2: TEdit;

Label2: TLabel;
DBGrid1: TDBGrid;
Label5: TLabel;
Edit8: TEdit;
Edit9: TEdit;
Label6: TLabel;
Edit4: TEdit;
Edit10: TEdit;
Button1: TButton;
Label3: TLabel;
Label11: TLabel;
Edit11: TEdit;
Label12: TLabel;
Edit12: TEdit;
Label15: TLabel;
Label16: TLabel;
Edit13: TEdit;
RadioButton4: TRadioButton;
RadioButton5: TRadioButton;
RadioButton7: TRadioButton;
DBGrid2: TDBGrid;
GroupBox1: TGroupBox;
Button2: TButton;
Label18: TLabel;
MaskEdit1: TMaskEdit;
MaskEdit2: TMaskEdit;
GroupBox2: TGroupBox;
Label19: TLabel;
Edit15: TEdit;
GroupBox3: TGroupBox;
Edit16: TEdit;
Button6: TButton;
DBGrid3: TDBGrid;

GroupBox5: TGroupBox;
GroupBox6: TGroupBox;
GroupBox7: TGroupBox;
GroupBox8: TGroupBox;
GroupBox9: TGroupBox;
GroupBox10: TGroupBox;
GroupBox11: TGroupBox;
GroupBox12: TGroupBox;
GroupBox13: TGroupBox;
GroupBox14: TGroupBox;
GroupBox16: TGroupBox;
GroupBox17: TGroupBox;
GroupBox20: TGroupBox;
GroupBox21: TGroupBox;
GroupBox22: TGroupBox;
GroupBox23: TGroupBox;
GroupBox18: TGroupBox;
GroupBox4: TGroupBox;
GroupBox15: TGroupBox;
GroupBox19: TGroupBox;
Label20: TLabel;
Label21: TLabel;
Label22: TLabel;
Label23: TLabel;
Label24: TLabel;
Label25: TLabel;
Label26: TLabel;
Label27: TLabel;
Label28: TLabel;
Label29: TLabel;
Label30: TLabel;
Label31: TLabel;
Label32: TLabel;

Label33: TLabel;
Label34: TLabel;
Label35: TLabel;
Label36: TLabel;
Label37: TLabel;
Label38: TLabel;
Label39: TLabel;
Button8: TButton;
Button7: TButton;
Label40: TLabel;
Label41: TLabel;
Label42: TLabel;
Label43: TLabel;
Label44: TLabel;
Label7: TLabel;
GroupBox24: TGroupBox;
o101: TPanel;
o201: TPanel;
o301: TPanel;
o401: TPanel;
o501: TPanel;
o502: TPanel;
o402: TPanel;
o302: TPanel;
o202: TPanel;
o102: TPanel;
o103: TPanel;
o203: TPanel;
o303: TPanel;
o403: TPanel;
o503: TPanel;
o104: TPanel;
o204: TPanel;

o304: TPanel;
o404: TPanel;
o504: TPanel;
Label54: TLabel;
Label53: TLabel;
Label52: TLabel;
Label51: TLabel;
Label50: TLabel;
Label48: TLabel;
Label47: TLabel;
Label46: TLabel;
Label45: TLabel;
Label49: TLabel;
Label17: TLabel;
Edit14: TEdit;
Edit17: TEdit;
Edit6: TEdit;
Edit1: TEdit;
Edit3: TEdit;
Button3: TButton;
adc1: TADOConnection;
Button5: TButton;

procedure Button4Click(Sender: TObject);
procedure Edit7Enter(Sender: TObject);
procedure ComboBox1Change(Sender: TObject);

procedure goster(Sender: TObject);

procedure FormCreate(Sender: TObject);
procedure Edit8Enter(Sender: TObject);
procedure DBGrid1CellClick(Column: TColumn);
procedure FormClose(Sender: TObject; var Action: TCloseAction);

```

procedure Edit2Change(Sender: TObject);
procedure Edit9Enter(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure RadioButton1Click(Sender: TObject);
procedure RadioButton1Enter(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure Edit15Change(Sender: TObject);
procedure TabbedNotebook1Change(Sender: TObject; NewTab: Integer;
    var AllowChange: Boolean);
procedure RadioButton4Click(Sender: TObject);
procedure RadioButton5Click(Sender: TObject);
procedure RadioButton7Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure Button8Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Edit16KeyPress(Sender: TObject; var Key: Char);
procedure MaskEdit2KeyPress(Sender: TObject; var Key: Char);
procedure o101MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o303MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure Edit8Exit(Sender: TObject);
procedure MaskEdit3Exit(Sender: TObject);
procedure o201MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o504MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o301MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o401MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o501MouseMove(Sender: TObject; Shift: TShiftState; X,

```



```

    Y: Integer);
procedure o102MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o202MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o302MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o402MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o502MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o103MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o203MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o403MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o503MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o104MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o204MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o304MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure o404MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure MaskEdit3KeyPress(Sender: TObject; var Key: Char);
procedure Edit2KeyPress(Sender: TObject; var Key: Char);
procedure Button3Click(Sender: TObject);
procedure Button5Click(Sender: TObject);

```

```

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

```

```

var
    Form1: TForm1;
    gonder:integer;
    cambaz:integer;

```

implementation

```

{$R *.dfm}
function fiyat_goster(oda:string):integer;
begin
    form1.adq1.close;
    form1.adq1.SQL.clear;
    form1.adq1.sql.add('select * from rooms where roomno=:oda');
    form1.adq1.parameters[0].Value:=oda;
    form1.adq1.Open;
    fiyat_goster:=form1.adq1.FieldName('unitprice').AsInteger;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
    ads1.CommandText:='select * from customers';
    ads1.Active:=true;
    ads1.Insert;
    ads1.FieldName('identityno').AsString:=maskedit3.Text;
    ads1.FieldName('name').AsString:=edit5.Text;
    ads1.FieldName('surname').AsString:=edit6.Text;
    ads1.FieldName('tel').AsString:=maskedit4.Text;

```

```

ads1.FieldName('active').AsBoolean:=true;
ads1.Post;
gonder:=ads1.FieldName('registerno').AsInteger;
ads1.Active:=false;

```

```

ads1.CommandText:='select * from custreg';
ads1.Active:=true;
ads1.Insert;
ads1.FieldName('registerno').AsInteger:=gonder;
ads1.FieldName('howmany').value:=combobox1.Text;
ads1.FieldName('roomno').AsString:=edit1.Text;
ads1.FieldName('enterdate').AsDateTime:=strtodate(edit7.Text);
ads1.Post;
ads1.Active:=false;

```

```

adq1.Close;
adq1.SQL.Clear;
adq1.SQL.Add('select * from rooms where roomno="'+edit1.Text+'" ');
adq1.Open;
adq1.Edit;
adq1.FieldName('empty').AsInteger:=adq1.fieldbyname('empty').AsInteger-
strtoint(combobox1.Text);
adq1.Post;

```

```

maskedit3.Text:='#####';
edit5.Text:="";
edit6.Text:="";
maskedit4.Text:=' ( ) - ';
edit7.Text:="";
combobox1.Text:='..choose';
edit1.Text:="";
end;

```



```
procedure TForm1.Edit7Enter(Sender: TObject);
```

```
begin
```

```
edit7.Text:=datetostr(date);
```

```
end;
```

```
procedure TForm1.ComboBox1Change(Sender: TObject);
```

```
begin
```

```
groupbox24.Visible:=true;
```

```
adq1.Close;
```

```
adq1.SQL.Clear;
```

```
adq1.SQL.Add('select * from rooms');
```

```
adq1.Open;
```

```
adq1.First;
```

```
repeat
```

```
Tpanel(Findcomponent('o'+adq1.fieldbyname('roomno').asString)).color:=clMenu;
```

```
adq1.Next;
```

```
until adq1.Eof;
```

```
adq1.Close;
```

```
adq1.SQL.Clear;
```

```
adq1.SQL.Add('select * from rooms where empty>=:bull');
```

```
adq1.Parameters[0].Value:=strtoint(combobox1.Text);
```

```
adq1.Open;
```

```
if adq1.RecordCount<>0 then begin
```

```
adq1.First;
```

```
repeat
```

```
Tpanel(Findcomponent('o'+adq1.fieldbyname('roomno').asString)).color:=clblue;
```

```
adq1.Next;
```

```
until adq1.Eof;
```

```
end
```

```
else showmessage('NOPE');
```

```
end;
```

```

procedure TForm1.goster(Sender: TObject);
var
  gelen:Tpanel;
begin
  gelen:=Tpanel(sender);
  adq1.Close;
  adq1.SQL.Clear;
  adq1.SQL.Add('select * from rooms where roomno=bul2');
  adq1.Parameters[0].Value:=copy(gelen.Caption,2,3);
  adq1.Open;
  if adq1.FieldByName('empty').value=0 then begin showmessage('this room is full');
  edit1.Text:="";
  edit17.text:="";
  end
  else
  edit1.text:=copy(gelen.Caption,2,4);
  edit14.Visible:=false;
  edit17.Visible:=true;
  edit17.text:=inttostr(adq1.FieldByName('unitprice').value);
  if (strtoint(adq1.FieldByName('capacity').AsString)<strtoint(combobox1.Text))
  then begin showmessage('THIS ROOM ISNT FOR YOU');
  edit1.Text:="";
  edit17.text:="";
  end;
  if
  (strtoint(adq1.FieldByName('capacity').AsString)>=(strtoint(combobox1.Text))) and
  (adq1.FieldByName('empty').value<strtoint(combobox1.Text)) then
  showmessage('THIS ROOM IS FOR YOU BUT FULL');
  end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  adc1.Connected:=true;

```

```

groupbox24.Visible:=false;
edit17.Visible:=false;
tabbednotebook1.PageIndex:=0;
end;
procedure TForm1.Edit8Enter(Sender: TObject);
begin
edit8.Text:=datetostr(date);
end;

procedure TForm1.DBGrid1CellClick(Column: TColumn);
begin

edit2.Text:=adq1.fieldbyname('name').AsString;
edit3.Text:=adq1.fieldbyname('surname').AsString;
edit4.Text:=adq1.fieldbyname('enterdate').Value;
edit10.Text:=adq1.fieldbyname('unitprice').Value;
edit11.Text:=adq1.fieldbyname('identityno').Value;
edit12.Text:=adq1.fieldbyname('howmany').Value;

cambaz:=adq1.fieldbyname('customers.registerno').Value;

end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
adc1.Connected:=false;
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
adq1.Close;
adq1.SQL.Clear;

```



```

adq1.SQL.Add('select      *      from      customers,custreg,rooms      where
rooms.roomno=custreg.roomno      and      customers.registerno=custreg.registerno      and
active=true and name like:a');
adq1.parameters[0].value:=edit2.text+'%';
adq1.Open;

```

```

end;

```

```

procedure TForm1.Edit9Enter(Sender: TObject);
begin

```

```

edit9.Text:=inttostr(strtoint(edit10.Text)*strtoint(edit12.Text)*strtoint(edit13.Text));
end;

```

```

procedure TForm1.Button2Click(Sender: TObject);

```

```

var k:integer;

```

```

begin

```

```

adq1.Close;

```

```

adq1.SQL.Clear;

```

```

adq1.SQL.Add('select      *      from      customers,custreg      where
(customers.registerno=custreg.registerno)      and      (custreg.enterdate>=t1)      and
(custreg.leavingdate<=t2)');

```

```

adq1.Parameters[0].Value:=maskedit1.text;

```

```

adq1.Parameters[1].Value:=maskedit2.text;

```

```

adq1.Open;

```

```

k:=0;

```

```

adq1.First;

```

```

while not adq1.Eof do begin

```

```

k:=k+1;

```

```

adq1.Next; end;

```

```

if k=0 then showmessage('THERE IS NO ANY REGISTRY BETWEEN THESE
DATES');

```

end;

procedure TForm1.RadioButton1Click(Sender: TObject);

begin

groupbox1.Visible:=true;

end;

procedure TForm1.RadioButton1Enter(Sender: TObject);

begin

groupbox1.Visible:=true;

end;

procedure TForm1.RadioButton2Click(Sender: TObject);

begin

groupbox2.Visible:=true;

end;

procedure TForm1.Button6Click(Sender: TObject);

begin

adq1.Close;

adq1.SQL.Clear;

adq1.SQL.Add('select * from customers,custreg where
(customers.registerno=custreg.registerno) and (custreg.odano="'+edit16.text+'")');

adq1.Open;

if adq1.RecordCount=0 then showmessage('UNTIL NOW NOBODY HAVE STAYED
IN THIS ROOM');

end;

procedure TForm1.Edit15Change(Sender: TObject);

begin

adq1.Close;

adq1.SQL.Clear;

```

adq1.SQL.Add('select      *      from      customers,custreg      where
customers.registerno=custreg.registerno and name like:a');
adq1.Parameters[0].Value:=edit15.Text+'%';
adq1.Open;

```

```

end;

```

```

procedure TForm1.TabbedNotebook1Change(Sender: TObject; NewTab: Integer;
var AllowChange: Boolean);
var i:integer;
begin
adq1.close;
adq1.SQL.Clear;
for i:=20 to 39 do
tlabel(findcomponent('label'+inttostr(i))).Caption:="";
radiobutton4.checked:=false;
radiobutton5.checked:=false;
radiobutton7.checked:=false;

end;

```

```

procedure TForm1.RadioButton4Click(Sender: TObject);
begin
adq1.Close;
adq1.SQL.Clear;
adq1.SQL.add('select      identityno,name,surname,roomno,howmany,enterdate      from
customers inner join custreg on customers.registerno=custreg.registerno where
active=true');
adq1.Open;
end;

```



```

procedure TForm1.RadioButton5Click(Sender: TObject);
begin
  adq1.Close;
  adq1.SQL.Clear;
  adq1.SQL.add('select identityno,name,surname,enterdate,leavingdate from customers
inner join custreg on customers.registerno=custreg.registerno where active=false');
  adq1.Open;
end;

```

```

procedure TForm1.RadioButton7Click(Sender: TObject);
begin
  adq1.Close;
  adq1.SQL.Clear;
  adq1.SQL.add('select
identityno,name,surname,active,roomno,howmany,enterdate,leavingdate      from
customers inner join custreg on customers.registerno=custreg.registerno');
  adq1.Open;
end;

```

```

procedure TForm1.Button7Click(Sender: TObject);
var i:integer;
begin
  adq1.Close;
  adq1.SQL.Clear;
  adq1.SQL.Add('select * from rooms');
  adq1.Open;
  i:=-1;
  adq1.First;
  while not adq1.Eof do begin
    i:=i+1;

```

```

TLabel(findcomponent('label'+inttostr(i+20))).caption:=inttostr(strtoint(adq1.fieldbyname('capacity')).asstring)-
adq1.fieldbyname('empty').asinteger)+'/'+adq1.fieldbyname('capacity').asstring;
adq1.Next;
end;

```

```

end;

```

```

procedure TForm1.Button8Click(Sender: TObject);

```

```

var i:integer;

```

```

begin

```

```

adq1.Close;

```

```

adq1.SQL.Clear;

```

```

adq1.SQL.Add('select * from rooms');

```

```

adq1.Open;

```

```

i:=-1;

```

```

adq1.First;

```

```

while not adq1.Eof do begin

```

```

i:=i+1;

```

```

TLabel(findcomponent('label'+inttostr(i+20))).caption:=inttostr(adq1.fieldbyname('empty').asinteger)+'/'+adq1.fieldbyname('capacity').asstring;

```

```

adq1.Next;

```

```

end;

```

```

end;

```

```

procedure TForm1.Button1Click(Sender: TObject);

```

```

begin

```

```

adq1.Close;

```

```

adq1.SQL.Clear;

```

```

adq1.SQL.Add('select * from rooms,customers,custreg where
rooms.roomno=custreg.roomno and customers.registerno=custreg.registerno and
customers.registerno=ol');

```

```

adq1.Parameters[0].Value:=cambaz;

```

```

adq1.Open;
adq1.Edit;
adq1.FieldName('leavingdate').AsDateTime:=strtodate(edit8.Text);
adq1.FieldName('empty').AsInteger:=adq1.FieldName('empty').AsInteger+strtoint
(edit12.text);
adq1.FieldName('active').AsBoolean:=False;
adq1.FieldName('total').Value:=strtoint(edit9.Text);
adq1.Post;
edit11.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
edit8.Text:="";
edit9.Text:="";
edit10.Text:="";
edit12.Text:="";
edit13.Text:="";

```

```

end;

```

```

procedure TForm1.Edit16KeyPress(Sender: TObject; var Key: Char);
begin
if key=#13 then begin
adq1.Close;
adq1.SQL.Clear;
adq1.SQL.Add('select      *      from      customers,custreg      where
(customers.registerno=custreg.registerno) and (custreg.roomno="'+edit16.text+'"');
adq1.Open;
if adq1.RecordCount=0 then showmessage('UNTIL NOW NOBODY HAVE STAYED
IN THIS ROOM..');
end;

```


end;

procedure TForm1.MaskEdit2KeyPress(Sender: TObject; var Key: Char);

var k:integer;

begin

if key=#13 then

begin

adq1.Close;

adq1.SQL.Clear;

adq1.SQL.Add('select * from customers,custreg where
(customers.registerno=custreg.registerno) and (custreg.enterdate>=t1) and
(custreg.leavingdate<=t2)');

adq1.Parameters[0].Value:=maskedit1.text;

adq1.Parameters[1].Value:=maskedit2.text;

adq1.Open;

k:=0;

adq1.First;

while not adq1.Eof do begin

k:=k+1;

adq1.Next; end;

if k=0 then showmessage('THERE IS NO REGISTRY BETWEEN THESE DATES');

end;

end;

procedure TForm1.o101MouseMove(Sender: TObject; Shift: TShiftState; X,

Y: Integer);

begin

edit14.Text:=inttostr(fiyat_goster('101'));

end;

```
procedure TForm1.o303MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    edit14.Text:=inttostr(fiyat_goster('303'));  
end;
```

```
procedure TForm1.Edit8Exit(Sender: TObject);  
begin  
    edit13.Text:=floattostr(strtodate(edit8.text)-adq1.fieldbyname('enterdate').AsDateTime);  
end;
```

```
procedure TForm1.MaskEdit3Exit(Sender: TObject);  
var  
    i,s:integer;  
begin  
    s:=0;  
    for i:=1 to length(maskedit3.Text) do begin  
        if copy(maskedit3.Text,i,1)<>'#' then s:=s+1;  
    end;  
    if s<11 then showmessage('ENTER MORE CHARACTER');  
end;
```

```
procedure TForm1.o201MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    edit14.Text:=inttostr(fiyat_goster('201'));  
end;
```

```
procedure TForm1.o504MouseMove(Sender: TObject; Shift: TShiftState; X,
```

```

    Y: Integer);
begin
edit14.Text:=inttostr(fiyat_goster('504'));

end;

procedure TForm1.o301MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
edit14.Text:=inttostr(fiyat_goster('301'));
end;

procedure TForm1.o401MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
edit14.Text:=inttostr(fiyat_goster('401'));
end;

procedure TForm1.o501MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
edit14.Text:=inttostr(fiyat_goster('501'));
end;

procedure TForm1.o102MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
edit14.Text:=inttostr(fiyat_goster('102'));
end;

procedure TForm1.o202MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin

```



```
edit14.Text:=inttostr(fiyat_goster('202'));  
end;
```

```
procedure TForm1.o302MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
edit14.Text:=inttostr(fiyat_goster('302'));  
end;
```

```
procedure TForm1.o402MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
edit14.Text:=inttostr(fiyat_goster('402'));  
end;
```

```
procedure TForm1.o502MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
edit14.Text:=inttostr(fiyat_goster('502'));  
end;
```

```
procedure TForm1.o103MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
edit14.Text:=inttostr(fiyat_goster('103'));  
end;
```

```
procedure TForm1.o203MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
edit14.Text:=inttostr(fiyat_goster('203'));  
end;
```

```
procedure TForm1.o403MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    edit14.Text:=inttostr(fiyat_goster('403'));  
end;
```

```
procedure TForm1.o503MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    edit14.Text:=inttostr(fiyat_goster('503'));  
end;
```

```
procedure TForm1.o104MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    edit14.Text:=inttostr(fiyat_goster('104'));  
end;
```

```
procedure TForm1.o204MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    edit14.Text:=inttostr(fiyat_goster('204'));  
end;
```

```
procedure TForm1.o304MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    edit14.Text:=inttostr(fiyat_goster('304'));  
end;
```

```
procedure TForm1.o404MouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin
```

```
edit14.Text:=inttostr(fiyat_goster('404'));  
end;
```

```
procedure TForm1.MaskEdit3KeyPress(Sender: TObject; var Key: Char);  
begin  
if key=char(VK_RETURN)then begin  
key:=#0;  
postmessage(Handle,WM_NEXTDLGCTL,0,0);  
end;  
end;
```

```
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);  
begin  
if key=char(VK_RETURN)then begin  
key:=#0;  
postmessage(Handle,WM_NEXTDLGCTL,0,0);  
end;  
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);  
begin  
edit11.Text:="";  
edit2.Text:="";  
edit3.Text:="";  
edit4.Text:="";  
edit8.Text:="";  
edit9.Text:="";  
edit10.Text:="";  
edit12.Text:="";  
edit13.Text:="";  
end;
```

```
procedure TForm1.Button5Click(Sender: TObject);
```

```
var  
a:integer;  
begin  
a:=messagedlg('ARE YOU SURE?',mtconfirmation,[mbytes, mbno],0);  
if a=mryes then  
application.Terminate;  
end;  
  
end.
```