NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Hospital Automation System Using By Delphi

Graduation Project
COM-400

Students: Yusuf ALAK (20041033)

Supervisor: Asist.Prof. Imanov ELBRUS

Nicosia – 2007

ACKNOWLEDGMENT

First of all, I would like to express my thanks to my supervisor Mr. Imanov Elbrus for supervising my project. Under the guidance of him I successfully overcome many difficulties. He welcomed me whenever I wanted to discuss something with him without feeling shame or hesitation Also I thank other instructors in Computer Engeenering department for their help and guideness.

Special thanks to my family, especially my parents for being patientfull during my undergraduate degree study. I could never have completed my study without their encouragement and endless support.

Finally, I want to thank all my friends and specially Metin Mescioğlu, Mustafa Arı ,Muharrem İlkuçar and Hasan Hüseyin Uysal who supported and helped me all the time.

TABLE OF CONTENTS

ABSTRACT

Hospital Automation System is going to prepare to solve problem that hospital processes which is made by hand. This system is going to make hospital processes fast, easily and more reliable.

The program provides manage and take hold of patient personal record, patient inspection records, patient laboratory records, patient cure and medicines records at a small or medium computer store. A scheduled user manual prepared for helping the users to select an suitable action.

Computer store automation system has different alternative user groups, these groups provide three choices of access availabilities, admins have more access availabilities that is whole program, accountants have restrictions on personnel area , users can access only products related menus.

Computer store automation system aims to help hospital primary doctor, hospital doctors and patients, the system provides easy, quick and more reliable process on company works.

INTRODUCTION

The hospitals were doing their processes manually, such as hold record of patient personal record, patient inspection records, patient's laboratory records, patient's cure and medicines records. But recently IT (Information Technology) started to help hospital. Then it has been very popular because it is faster, cheaper than manually and so easy work with IT.

It is necessary to hospital work with computer programs in their work to more valuable present and more efficiently working. Doctors, users and patients feel their self they are really in a technological company when they work or being in a program like this.

At this point Hospital Automation System will provide easiness and quickness of hospital processes that are of patient personal record, patient inspection records, patient laboratory records, patient cure and medicines records. Also system has a authorization steps to determine and control the access levels, to use this feature every user have different authorization levels that given by managers with username and password.

CHAPTER ONE

1. BASIC CONCEPT OF DELPHI 6

1.1 Introduction

The name "Delphi" was never a term with which either Olaf Helmer or Norman Dalkey (the founders of the method) were particular happy. Since many of the early Delphi studies focused on utilizing the technique to make forecasts of future occurrences, the name was first applied by some others at Rand as a joke. However, the name stuck. The resulting image of a priestess, sitting on a stool over a crack in the earth, inhaling sulfur fumes, and making vague and jumbled statements that could be interpreted in many different ways, did not exactly inspire confidence in the method.

The straightforward nature of utilizing an iterative survey to gather information "sounds" so easy to do that many people have done "one" Delphi, but never a second. Since the name gives no obvious insight into the method and since the number of unsuccessful Delphi studies probably exceeds the successful ones, there has been a long history of diverse definitions and opinions about the method. Some of these misconceptions are expressed in statements such as the following that one finds in the literature:

- It is a method for predicting future events.
- It is a method for generating a quick consensus by a group.
- It is the use of a survey to collect information.
- It is the use of anonymity on the part of the participants.
- It is the use of voting to reduce the need for long discussions.
- It is a method for quantifying human judgement in a group setting.

Some of these statements are sometimes true; a few (e.g. consensus) are actually contrary to the purpose of a Delphi. Delphi is a communication structure aimed at producing detailed critical examination and discussion, not at forcing a quick compromise. Certainly quantification is a property, but only to serve the goal of quickly identifying agreement and disagreement in order to focus attention. It is often very common, even today, for people to come to a view of the Delphi method that reflects a particular

application with which they are familiar. In 1975 Linstone and Turoff proposed a view of the Delphi method that they felt best summarized both the technique and its objective.

The essence of Delphi is structuring of the group communication process. Given that there had been much earlier work on how to facilitate and structure face-to-face meetings, the other important distinction was that Delphi was commonly applied utilizing a paper and pencil communication process among groups in which the members were dispersed in space and time. Also, Delphis were commonly applied to groups of a size (30 to 100 individuals) that could not function well in a face-to-face environment, even if they could find a time when they all could get together.

The result, however, is not merely confusion due to different names to describe the same things; but a basic lack of knowledge by many people working in these areas as to what was learned in the studies of the Delphi Method about how to properly employ these techniques and their impact on the communication process. There seems to be a great deal of "rediscovery" and repeating of earlier misconceptions and difficulties.

Given this situation, the primary objective of this chapter is to review the specific properties and methods employed in the design and execution of Delphi Exercises and to examine how they may best be translated into a computer based environment.

1.2 What is Delphi?

Delphi is an object-oriented, visual programming environment for rapid application development (RAD). With Delphi, you can write Windows programs more quickly and more easily than was ever possible before. You can create Win32 console applications or Win32 graphical user interface (GUI) programs. When creating Win32 GUI applications with Delphi, you have all the power of a true compiled programming language (Object Pascal) wrapped up in a RAD environment. What this means is that you can create the user interface to a program (the user interface means the menus, dialog boxes, main window, and so on) using drag-and-drop techniques for true rapid application development. You can also drop ActiveX controls on forms to create specialized programs such as Web browsers in a matter of minutes. Delphi gives you all this, and at virtually no cost: You don't sacrifice program execution speed because Delphi generates fast compiled code. Delphi provides all the tools you need to develop, test, and deploy applications, including a large library of

reusable components, a suite of design tools, application and form templates, and programming wizards. Delphi does a good job of hiding some of the low-level details that make up the guts of a Windows program, but it cannot write programs for you. In the end, you must still be a programmer, and that means you have to learn programming. That can be a long, uphill journey some days. The good news is that Delphi can make your trek fairly painless and even fun. Yes, you can work and have fun doing it!

1.2.1 Developer Support Services and Web Site

Borland offers a variety of support options to meet the needs of its diverse developer community. To find out about support, refer to http://www.borland.com/devsupport/. From the Web site, you can access many newsgroups where Delphi developers exchange information, tips, and techniques. From the Web site, you can access many newsgroups where Delphi developers exchange information, tips, and techniques. The site also includes a list of books about Delphi, additional Delphi technical documents, and Frequently Asked Questions (FAQs).

1.3 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).

1.3.1 Starting Delphi

You can start Delphi in the following ways:

- Double-click the Delphi icon (if you've created a shortcut).
- Choose Programs Borland Delphi 6|Delphi 6 from the Windows Start menu.
- Choose Run from the Windows Start menu, then enter Delphi32.
- Double-click Delphi32.exe in the Delphi\Bin directory.

1.3.2Delphi (IDE)

When you first start Delphi, you'll see some of the major tools in the IDE. In Delphi, the IDE includes the menus, toolbars, Component palette, Object Inspector, Object TreeView, Code editor, Code Explorer, Project Manager, and many other tools. The particular features and components available to you will depend on which edition of Delphi you've purchased.

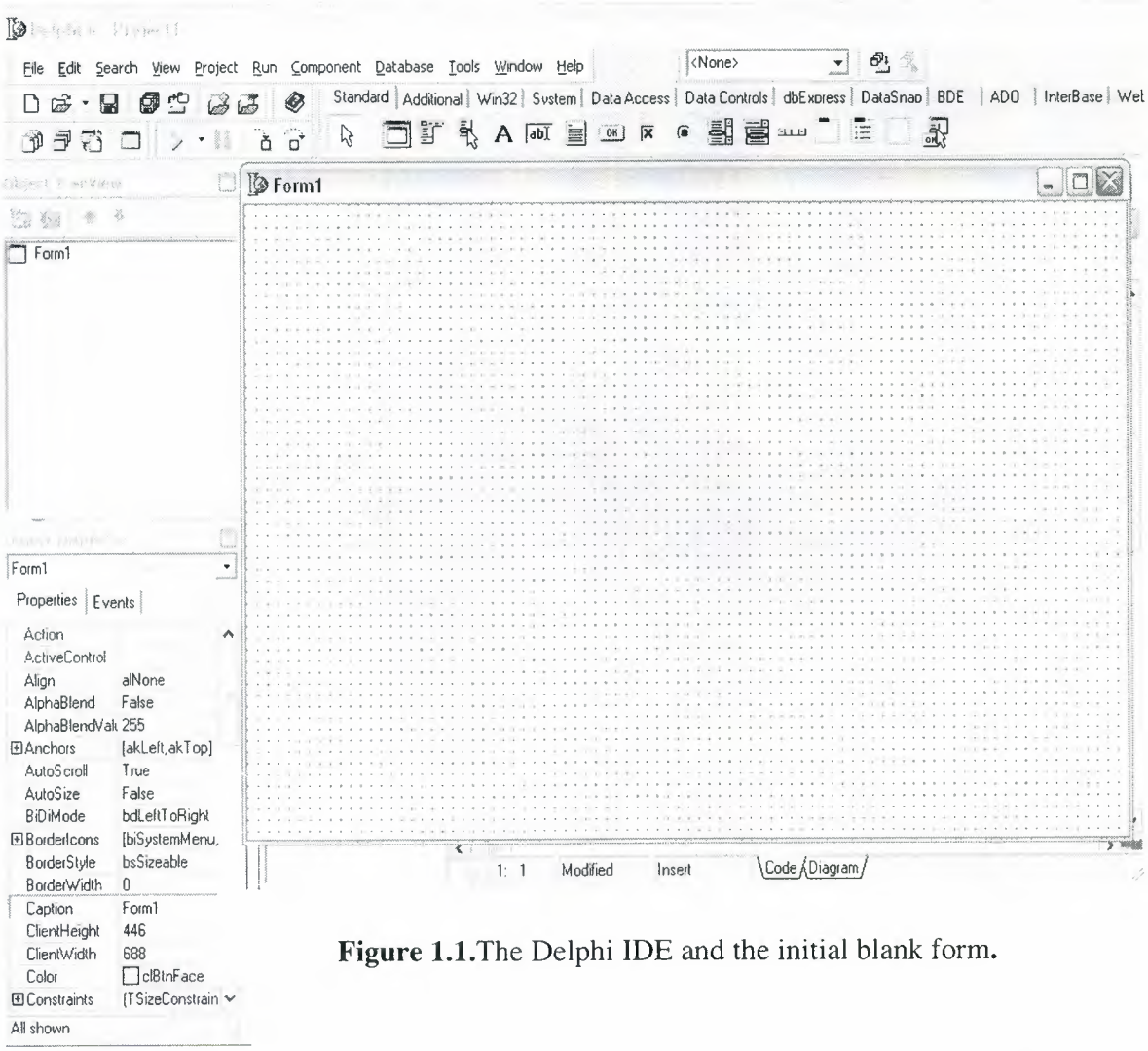


Figure 1.1. The Delphi IDE and the initial blank form.

The Delphi IDE is divided into three parts. The top window can be considered the main window. It contains the toolbars and the Component palette. The Delphi toolbars give you one-click access to tasks such as opening, saving, and compiling projects. The Component palette contains a wide array of components that you can drop onto your forms.

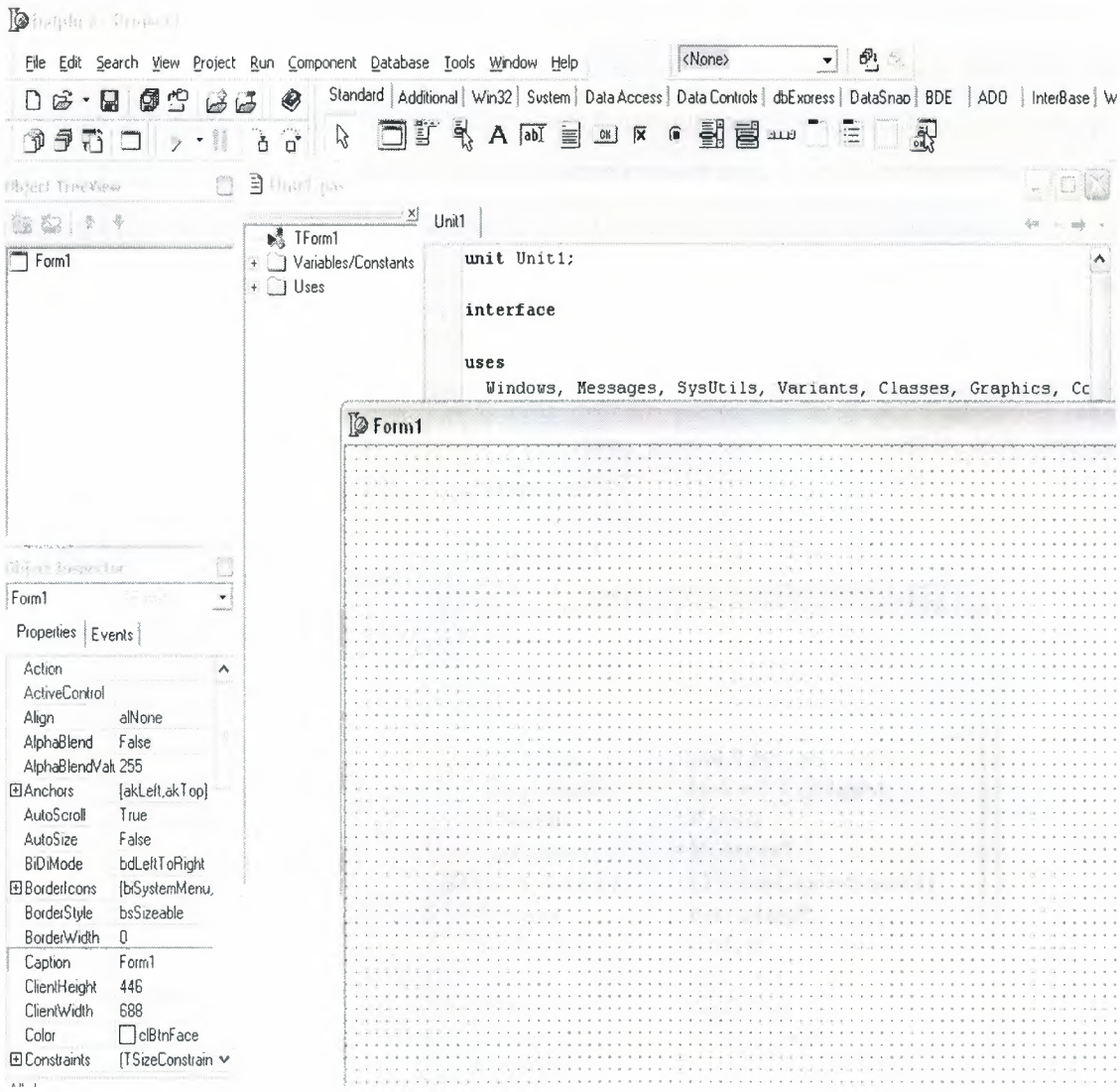


Figure. 1.2 IDE

Delphi's development model is based on two-way tools. This means that you can move back and forth between visual design tools and text-based code editing. For example, after using the Form Designer to arrange buttons and other elements in a graphical interface, you can immediately view the form file that contains the textual description of your form. You can also manually edit any code generated by Delphi without losing access to the visual programming environment.

Delphi's development model is based on two-way tools. This means that you can move back and forth between visual design tools and text-based code editing. For example, after using the Form Designer to arrange buttons and other elements in a graphical interface, you can immediately view the form file that contains the textual description of your form. You can also manually edit any code generated by Delphi without losing access to the visual programming environment.

From the IDE, all your programming tools are within easy reach. You can design graphical interfaces, browse through class libraries, write code, and compile, test, debug, and manage projects without leaving the IDE.

1.3.2 The Object Inspector

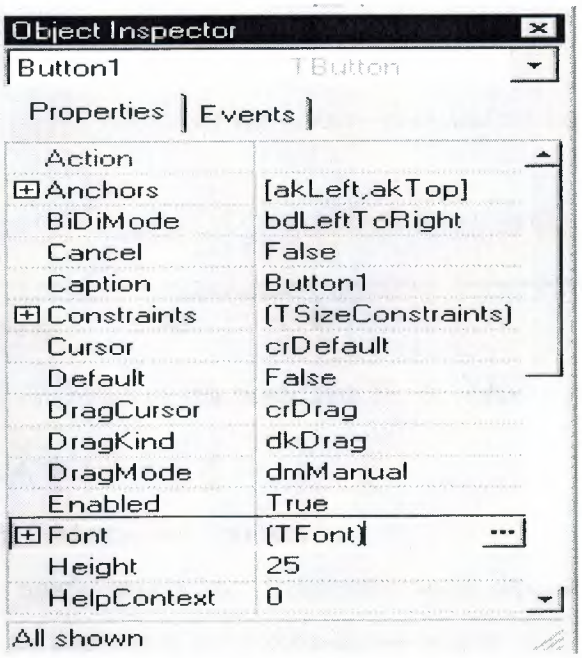


Figure. 1.3 object inspector

Below the main window and on the left side of the screen is the Object Inspector. It is through the Object Inspector that you modify a component's properties and events. You will use the Object Inspector constantly as you work with Delphi. The Object Inspector has two tabs: the Properties tab and the Events tab. A component's properties control how the component operates. For example, changing the Color property of a component changes the background color of that component.

1.3.4 The Delphi Workspace

The main part of the Delphi IDE is the workspace. The workspace initially displays the Form Designer. It should come as no surprise that the Form Designer enables you to create forms. In Delphi, a form represents a window in your program. The form might be the program's main window, a dialog box, or any other type of window. You use the Form Designer to place, move, and size components as part of the form creation process.

Hiding behind the Form Designer is the Code Editor. The Code Editor is where you type code when writing your programs. The Object Inspector, Form Designer, Code Editor, and Component palette work interactively as you build applications.

Now that you've had a look at what makes up the Delphi IDE, let's actually do something.

1.3.5 The Menus and Toolbars

The main window, which occupies the top of the screen, contains the main menu, toolbars, and Component palette.

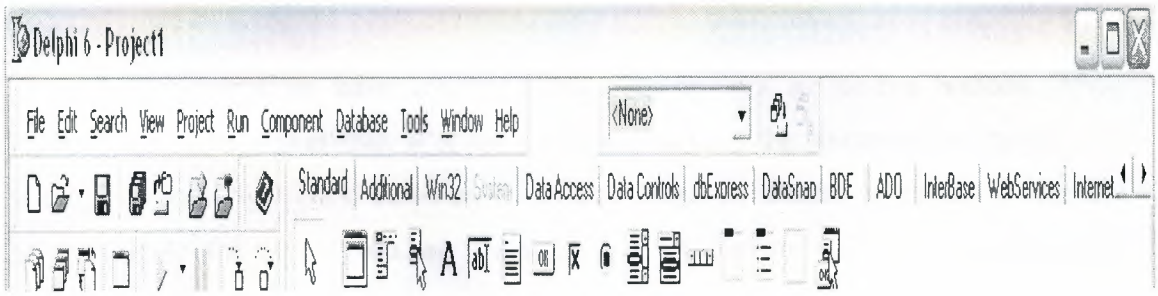


Figure. 1.4 Menus and Toolbars

Delphi's toolbars provide quick access to frequently used operations and commands. Most toolbar operations are duplicated in the drop-down menus.

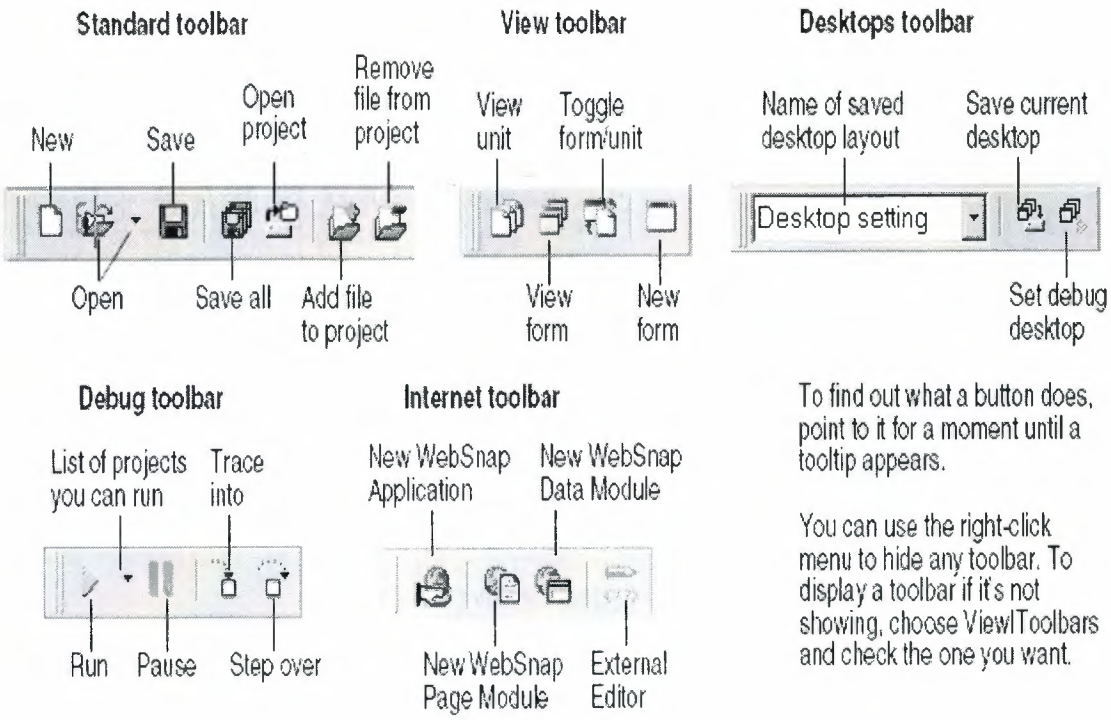


Figure. 1.5 Toolbars

Many operations have keyboard shortcuts as well as toolbar buttons. When a keyboard shortcut is available, it is always shown next to the command on the dropdown menu. You can right-click on many tools and icons to display a menu of commands appropriate to the object you are working with. These are called context menus.The toolbars are also customizable. You can add commands you want to them or move them to different locations.

1.3.6 The Component Palette and Form Designer

The Component palette, Form Designer, Object Inspector, and Object TreeView work together to help you build a user interface for your application. The Component palette includes tabbed pages with groups of icons representing visual or nonvisual components. The pages divide the components into various functional groups. For example, the Standard, Additional, and Win32 pages include windows controls such as an edit box and up/down button; the Dialogs page includes common dialog boxes to use for file operations such as opening and saving files.

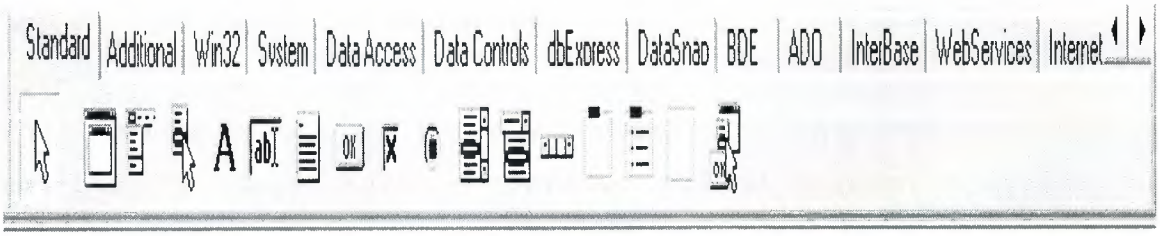


Figure. 1.6 Component Palatte

Each component has specific attributes properties, events, and methods that enable you to control your application. After you place components on the form, or Form Designer, you can arrange components the way they should look on your user interface.

After you place components on a form, the Abject inspector dynamically changes the set of properties

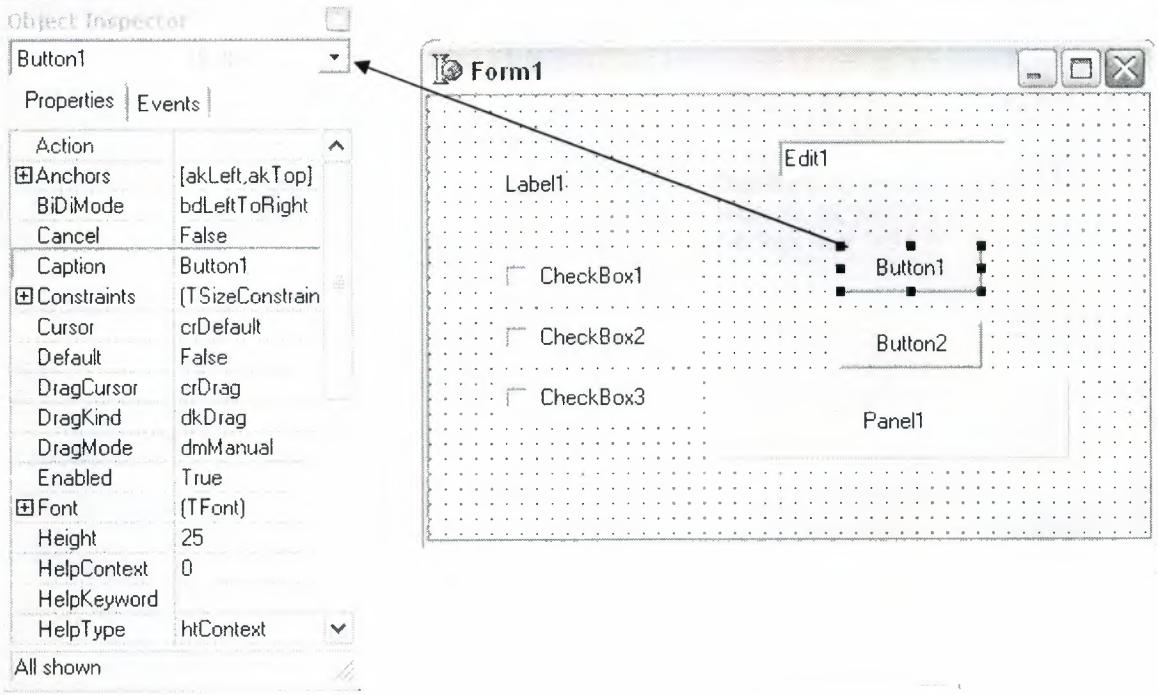


Figure. 1.7 Changing Set of Properties in Object Inspector

1.3.7 The Object Tree View

The Object Tree View displays a component's sibling and parent-child relationships in a hierarchical, or tree diagram. The tree diagram is synchronized with the Object

Inspector and the Form Designer so that when you change focus in the Object Tree View, both the Object Inspector and the form change focus.

You can use the Object Tree View to change related components' relationships to each other. For example, if you add a panel and check box component to your form, the two components are siblings. But in the Object Tree View, if you drag the check box on top of the panel icon, the check box becomes the child of the panel.

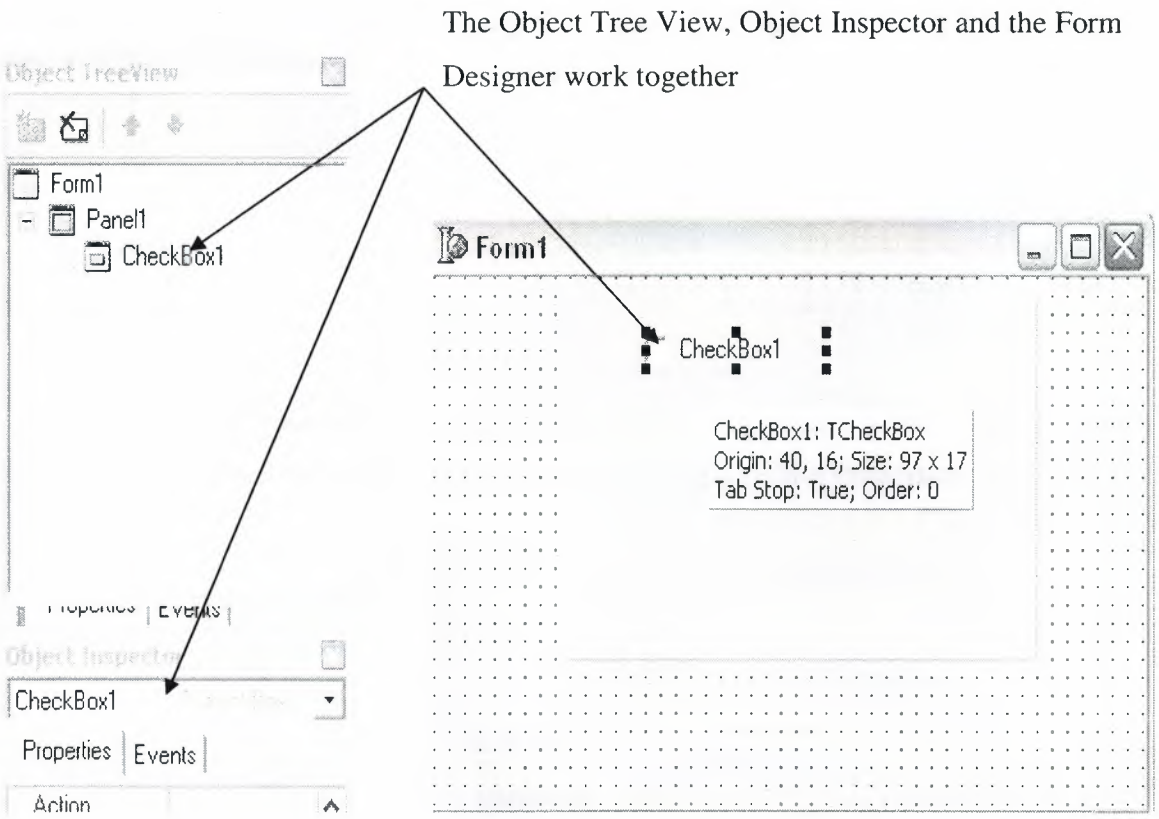


Figure 1.8 Panel

The Object Tree View is especially useful for displaying the relationships between database objects.

1.3.8 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 | Other to display the New Items dialog box when you begin a project.

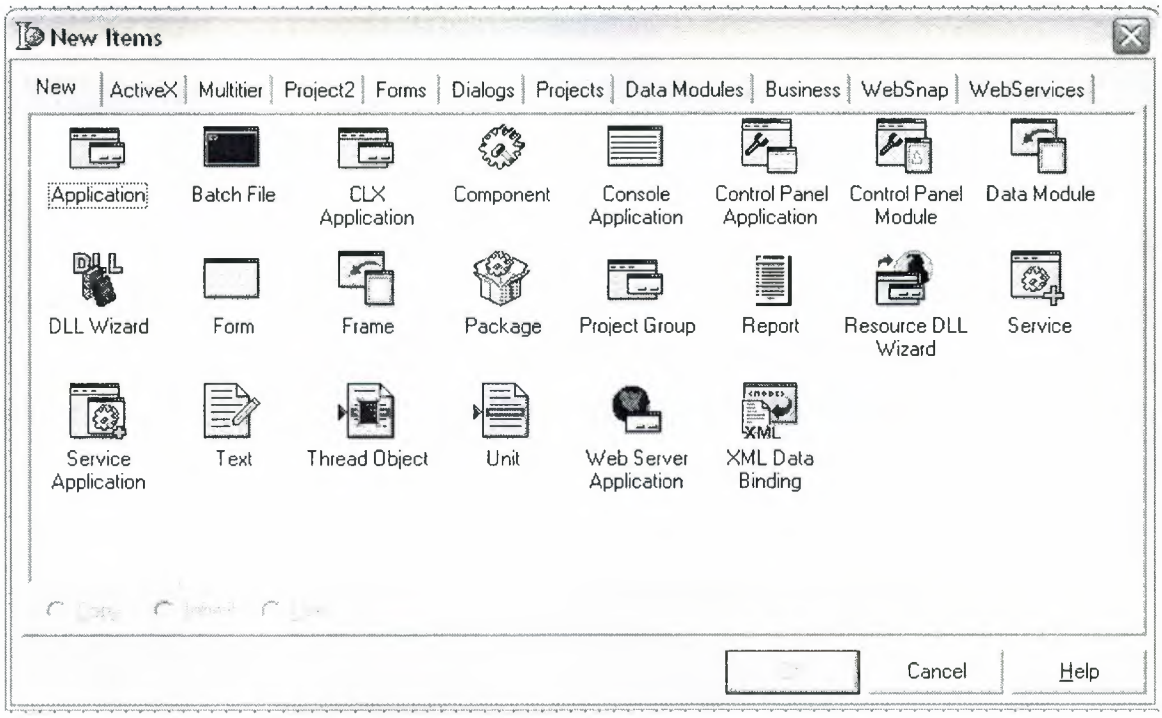


Figure. 1.9 Object Repository

To edit or remove objects from the Object Repository, either choose Tools | Repository or right-click in the New Items dialog box and choose Properties.

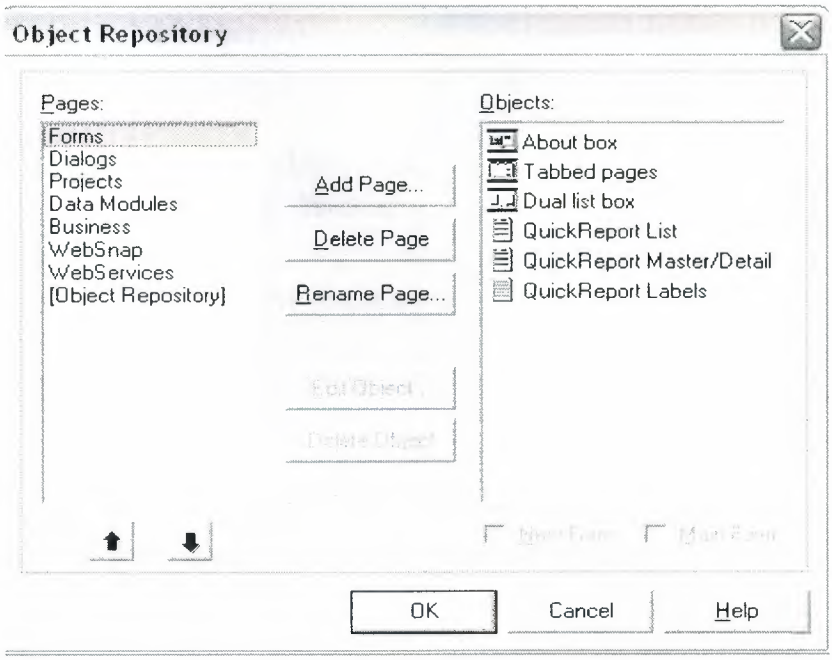


Figure. 1.10 Adding project and form templates to the Object Repository

1.3.9 The Code Editor

As you design the user interface for your application, Delphi generates the underlying Delphi code. When you select and modify the properties of forms and objects, your changes are automatically reflected in the source files. You can add code to your source files directly using the built-in Code editor, which is a full-featured ASCII editor. Delphi provides various aids to help you write code, including the Code Insight tools, class completion, and code browsing.

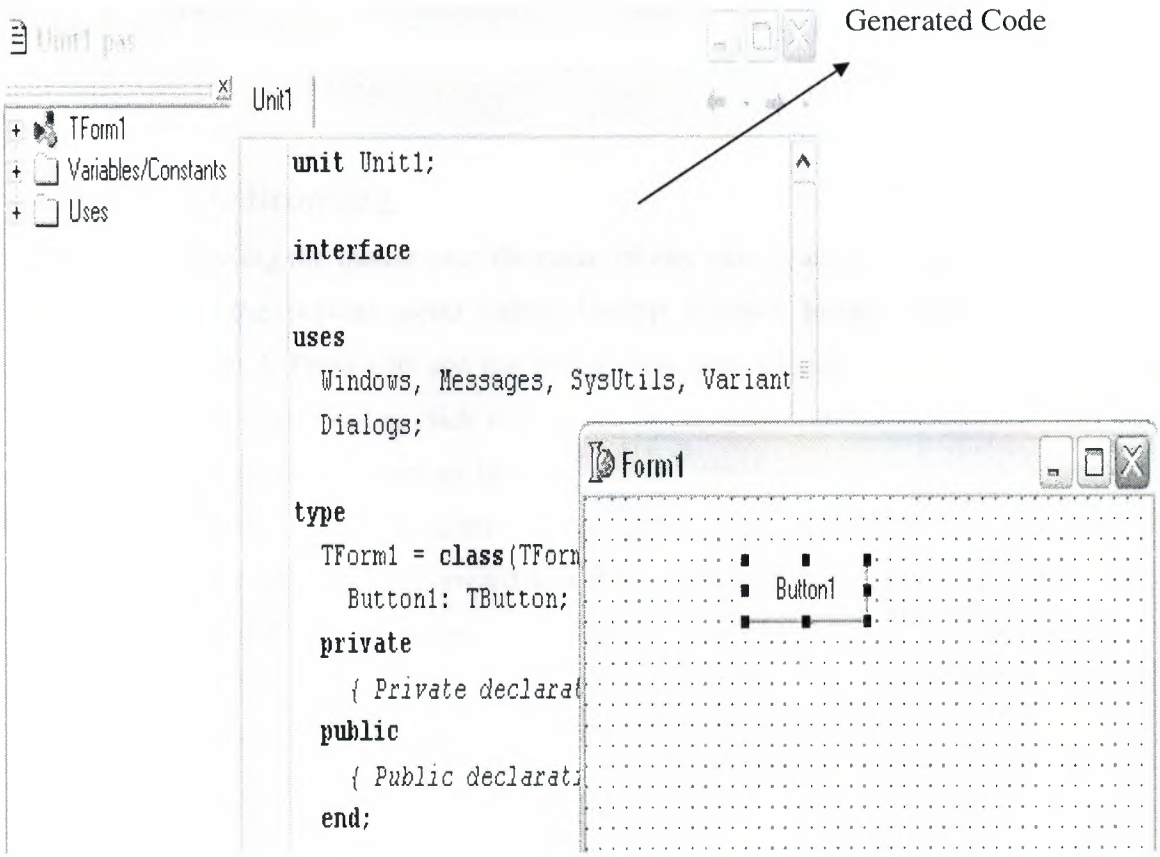


Figure. 1.11 Code Editor

The Code Insight tools display context-sensitive pop-up windows.

To turn these tools on or off, choose Tools | Editor Options and click the Code Insight tab. Check or uncheck the tools in the Automatic features section.

1.3.10 Class Completion

Class completion generates skeleton code for classes. Place the cursor anywhere within a class declaration of the interface section of a unit and press Ctrl+Shift+C or right click and choose Complete Class at Cursor. Delphi automatically adds private read and write specifiers to the declarations for any properties that require them, then creates keleton 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 turn on class completion, choose Tools Environment Options, click the Explorer tab, and make sure Finish incomplete properties is checked.

1.3.11 Code Browsing

While passing the mouse over the name of any class, variable, property, method, or other identifier, the pop-up menu called Tooltip Symbol Insight displays where the identifier is declared. Press Ctrl and the cursor turns into a hand, the identifier turns blue and is underlined, and you can click to jump to the definition of the identifier. The Code editor has forward and back buttons like the ones on Web browsers. As you jump to these definitions, the Code editor keeps track of where you've been in the code. You can click the drop-down arrows next to the Forward and Back buttons to ove forward and backward through a history of these references.

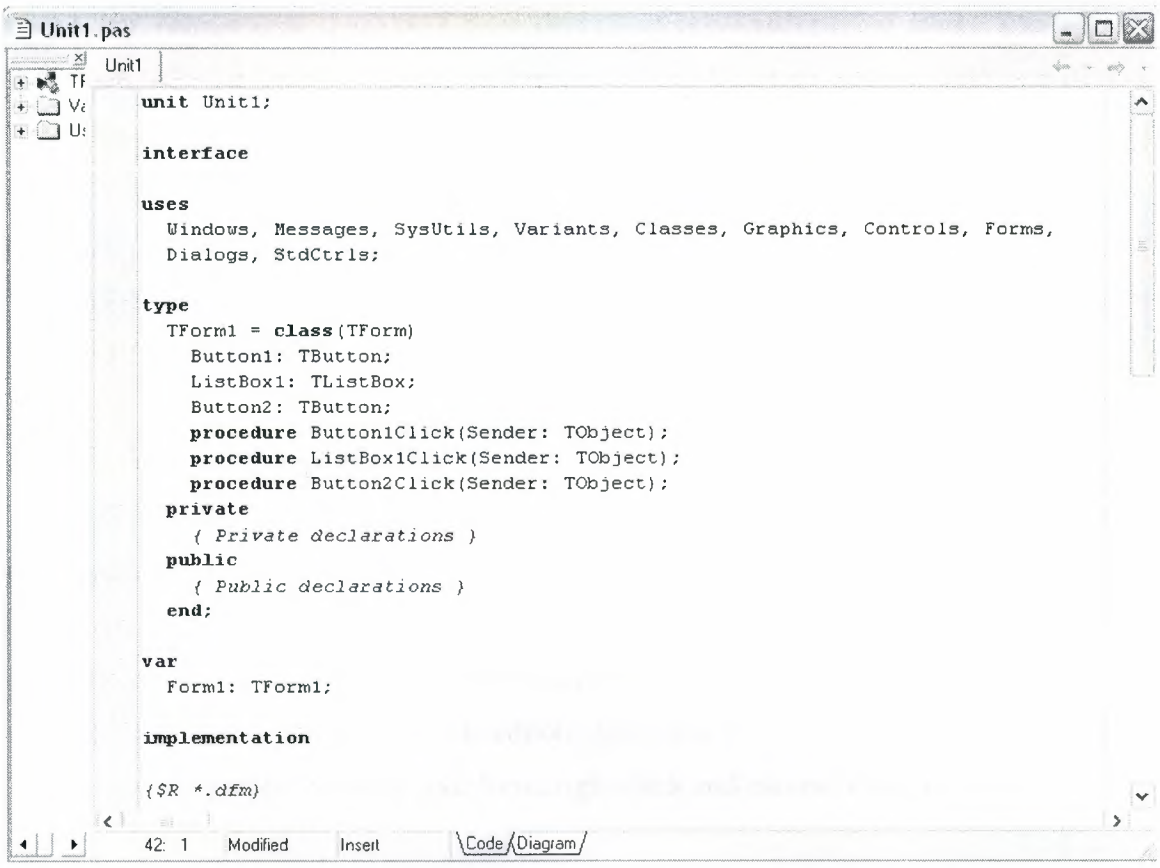


Figure. 1.12 Code Editor

You can also move between the declaration of a procedure and its implementation by pressing Ctrl+Shift+↑ or Ctrl+Shift+↓.

To customize your code editing environment, see "Customizing the Code Editor".

1.3.12 The Diagram Page

The bottom of the Code editor may contain one or more tabs, depending on which edition of Delphi you have. The Code page, where you write all your code, appears in the foreground by default. The Diagram page displays icons and connecting lines representing the relationships between the components you place on a form or data module. These relationships include siblings, parent to children, or components to properties. To create a diagram, click the Diagram page.

For components that don't have dependent relationships but where you want to show one, use the toolbar buttons at the top of the Diagram page to add one of four connector types, including allude, property, master/detail, and lookup. You can also add comment blocks that connect to each other or to a relevant icon.

You can type a name and description for your diagram, save the diagram, and print it when you are finished

.

1.3.13 Wiewing Form Code

Forms are a very visible part of most Delphi projects they are where you design the user interface of an application. Normally, you design forms using Delphi's visual tools, and Delphi stores the forms in form files. Form files (.dfm, or .xfm for a CLX application) describe each component in your form, including the values of all persistent properties. To view and edit a form file in the Code editor, right-click the form and select View as Text. To return to the graphic view of your form, right-click and choose View as Form.

You can save form files in either text (the default) or binary format. Choose Tools| Environment Options, click the Designer page, and check or uncheck the New forms as text check box to designate which format to use for newly created forms.

1.3.14 The Code Explorer

When you open Delphi, the Code Explorer is docked to the left of the Code editor window, depending on whether the Code Explorer is available in the edition of Delphi you have. The Code Explorer displays the table of contents as a tree diagram for the source code open in the Code editor, listing the types, classes, properties, methods, global variables, and routines defined in your unit. It also shows the other units listed in the uses clause.

You can use the Code Explorer to navigate in the Code editor. For example, if you double-click a method in the Code Explorer, a cursor jumps to the definition in the class declaration in the interface part of the unit in the Code editor.

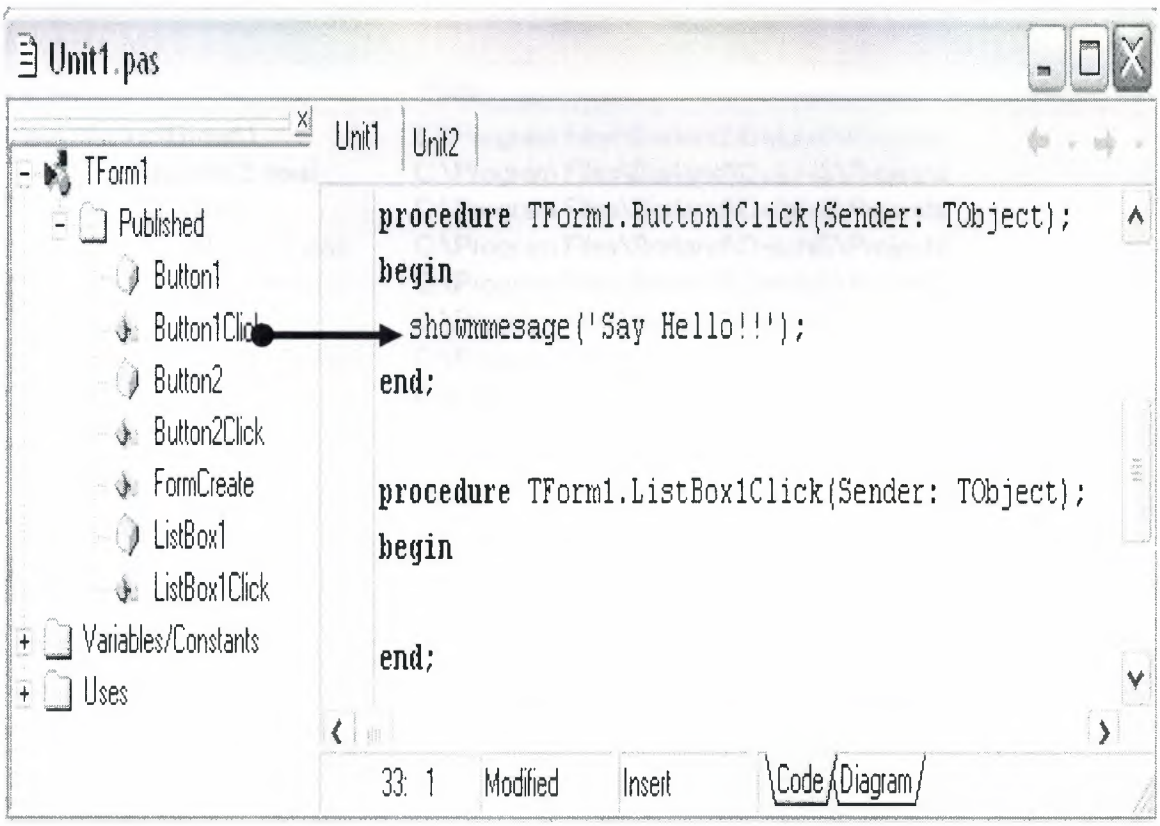


Fig. 1.13 Code Explorer

To configure how the Code Explorer displays its contents, choose Tools|Environment Options and click the Explorer tab.

1.3.15 The Project Manager

When you first start Delphi, it automatically opens a new project. A project includes several files that make up the application or DLL you are going to develop. You can view and organize these files such as form, unit, resource, object, and library files in a project management tool called the Project Manager. To display the Project Manager, choose View | Project Manager.

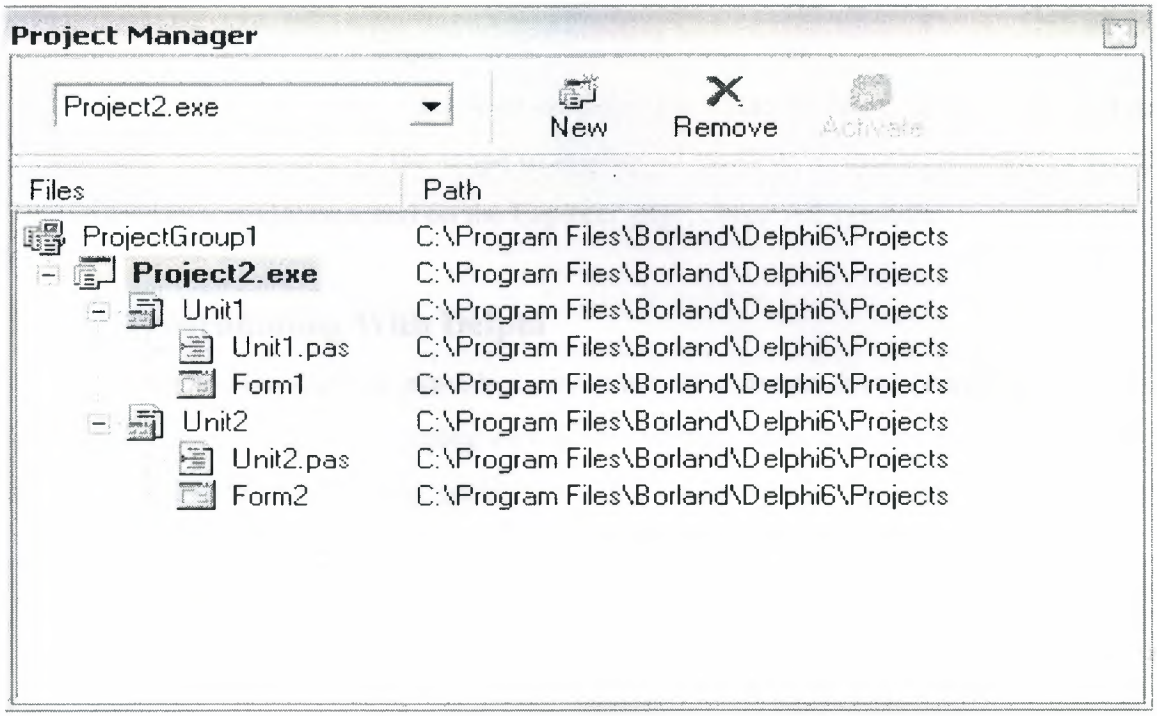


Figure. 1.14 Project Manager

1.3.16 The Project Browser

The Project Browser examines a project in detail. The Browser displays classes, units, and global symbols (types, properties, methods, variables, and routines) your project declares or uses in a tree diagram. Choose View Browser to display the Project Browser.

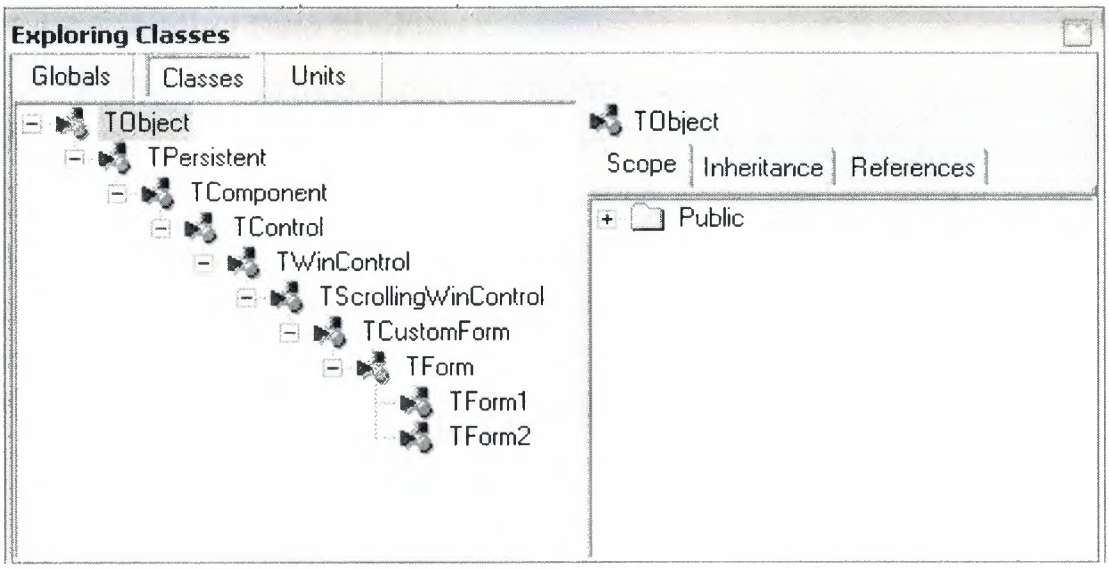


Figure. 1.15 Project Browser

By default, the Project Browser displays the symbols from units in the current project only. You can change the scope to display all symbols available in Delphi. Choose Tools Environment Options, and on the Explorer page, check All symbols.

1.4 Programming With Delphi

The following sections provide an overview of software development with Delphi, including creating a project, working with forms, writing code, and compiling, debugging, deploying, and internationalizing applications, and including the types of projects you can develop.

1.4.1 Creating a Project

A project is a collection of files that are either created at design time or generated when you compile the project source code. When you first start Delphi, a new project opens. It automatically generates a project file (Project1.dpr), unit file (Unit1.pas), and resource file (Unit1.dfm; Unit1.xfm for CLX applications), among others. If a project is already open but you want to open a new one, choose either File | New | Application or File | New | Other and double-click the Application icon.

1.4.2 Adding Data Modules

A data module is a type of form that contains nonvisual components only. Nonvisual components can be placed on ordinary forms alongside visual components. But if you plan on reusing groups of database and system objects, or if you want to isolate the parts of your application that handle database connectivity and business rules, data modules provide a convenient organizational tool.To create a data module, choose File|New|Data Module.

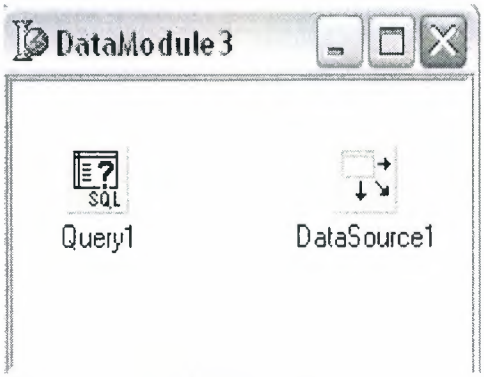


Figure. 1.16 Adding Data Modules

When you reopen an existing data module, Delphi displays its components.

1.4.3 Building the user interface

With Delphi, you first create a user interface (UI) by selecting components from the Component palette and placing them on the main form.

1.4.4 Placing components on a form

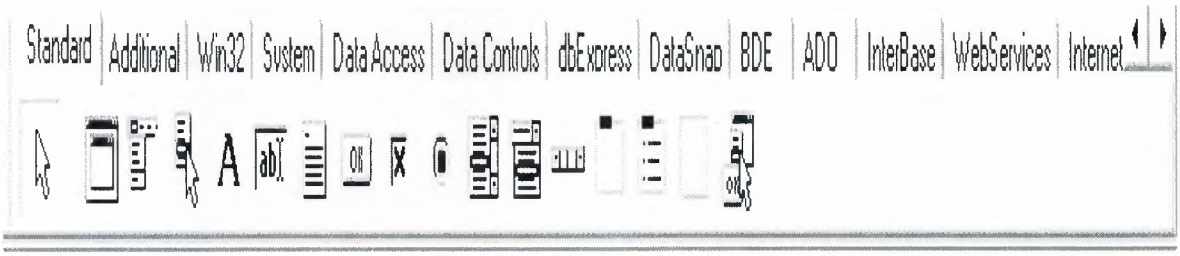


Figure. 1.17 Component Palette

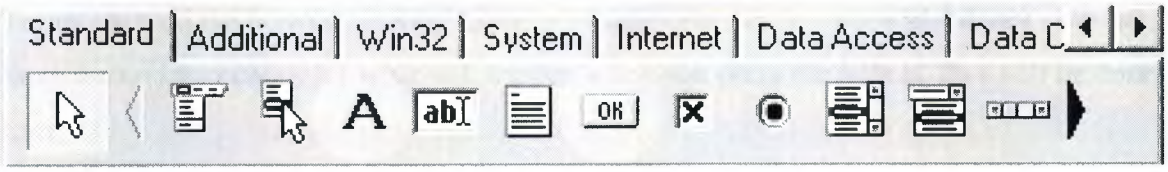


Figure.1.18 Placing Component

To place a Button on the form, click once on the Button component on the toolbar. Then move the mouse cursor over to the Form and click on the Form where you want the Button to be. Repeat the same procedure with the Label component.

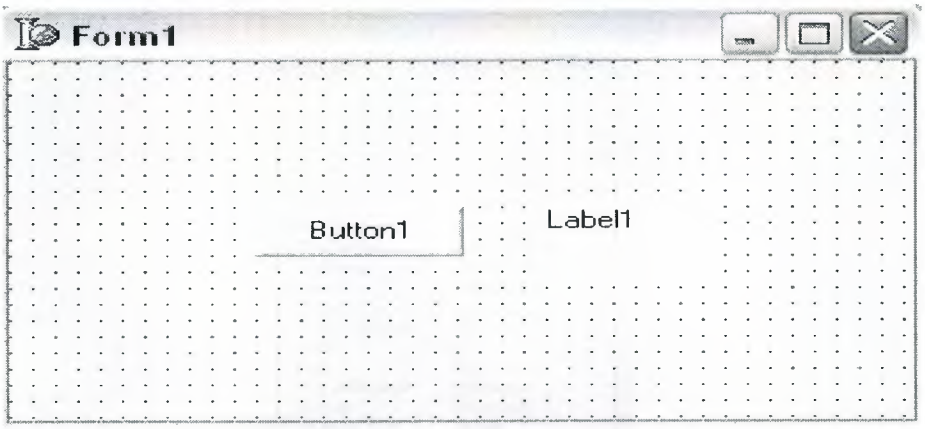


Figure.1.19 Buton and Label

It will look something like this. Actually you can run your application now. Simply press F9, click Ok to save your project and it's running.

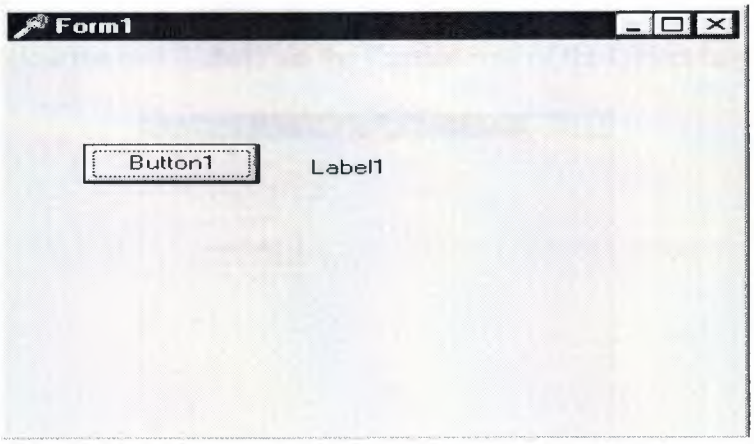


Figure.1.20. Run

Try to click on the button. Nothing happens? Well, since we have not added any code yet, there are no instructions for what will happen when you press the button. This will be done later.

1.4.5 Setting the properties of the components

To change the text 'Button1' on the button, we change the value of its Caption property. Click on the Button once ('select' the Button), move to the Object Inspector and enter the new value on the Caption row.

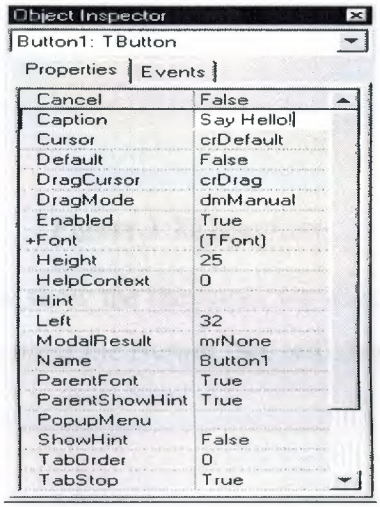


Figure.1.21. Properties Component

To remove the text from the Label, we set the Caption property to '' (empty string). Select the Label and delete the text 'Label1' on the Caption row of the Object Inspector.

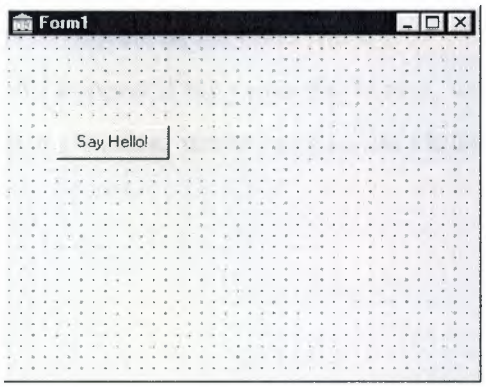


Figure.1.22. Button (Say Hello)

If we run our application now, we see that we have made some progress since the first step. Press F9 to run.

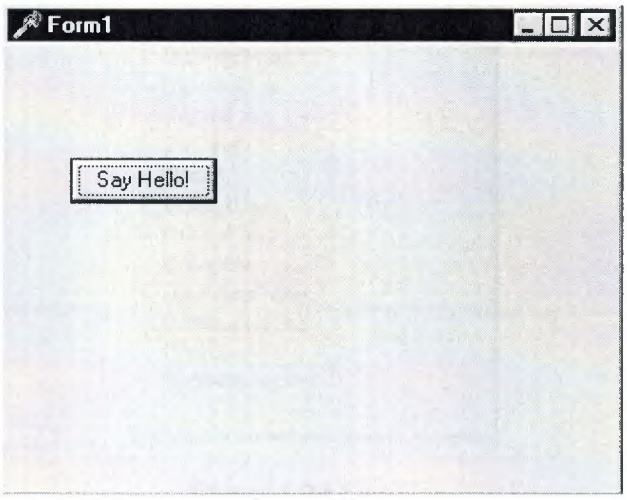


Figure.1.23. Run form

The Button and the Label now show the messages we want them to do. However, it still doesn't happen anything if we press the Button. We will deal with this now.

1.4.6 Writing Code

An integral part of any application is the code behind each component. While Delphi's RAD environment provides most of the building blocks for you, such as preinstalled visual and nonvisual components, you will usually need to write event handlers, methods, and perhaps some of your own classes. To help you with this task, you can choose from thousands of objects in the class library.

To specify what will happen if we press the button, we enter code for the OnClick event of the Button. Select the button, move over to the Object Inspector and click on the Events tab.

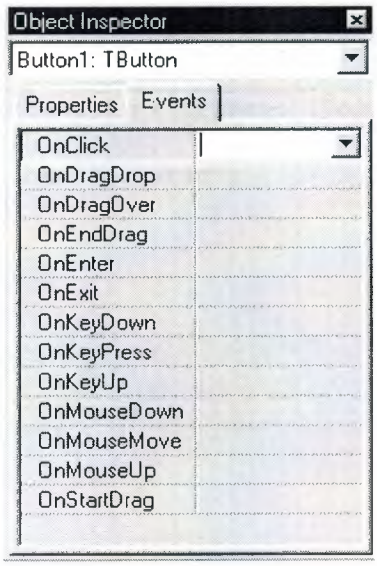


Figure1.24. OnClick

Double click on the OnClick row. The Code Editor will popup and a procedure for the OnClick event will be created.

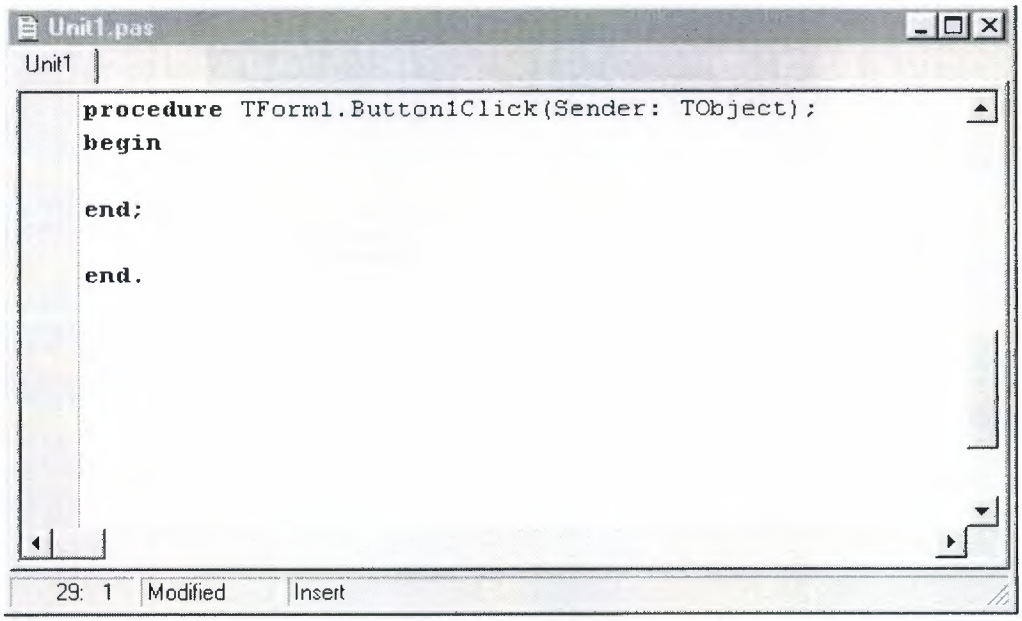


Figure.1.25. Unit.pas page

You can fill this empty procedure with code that is going to be executed when you press the Button. The task of this application was to let the Label show the 'Hello World' message when we click on the button, remember?

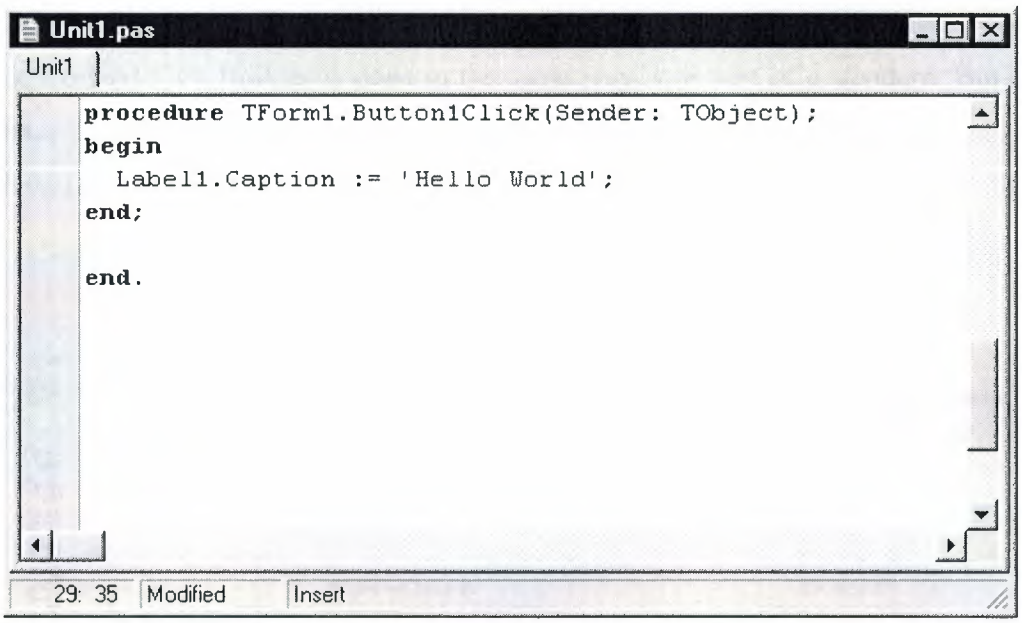


Figure.1.26. On Cick Operation

This code should do the trick. It will be explained in the next lessons. Just type it in, run the application and press the Button.

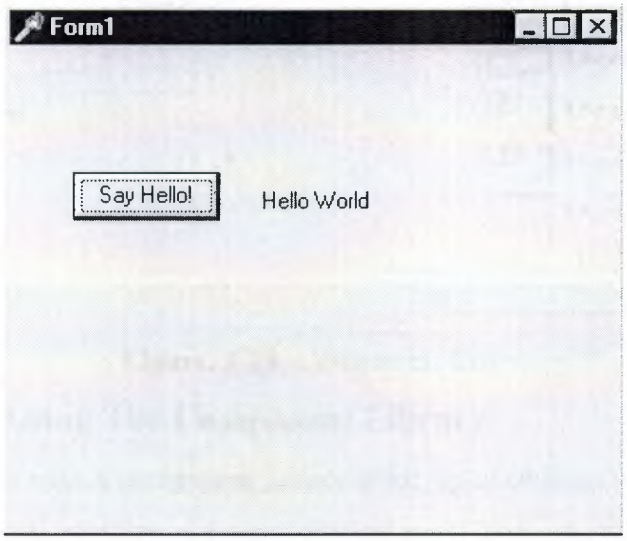


Figure.1.27. Run Aplication (say hello)

Here we go! Hello World!

As you have noticed by now, this is not a very useful application. However, by creating it you have learnt the three essential steps of creating any application in Borland

Delphi. Creating applications in other 'visual' programming languages like Visual Basic or Borland C++ Builder is done in the same way, it is sort of a standard. But once you have learnt some more Delphi, you will probably not change to another programming language unless you have to. Delphi has all the features you will ever need.

Select the component and drag it to wherever you want on the form.

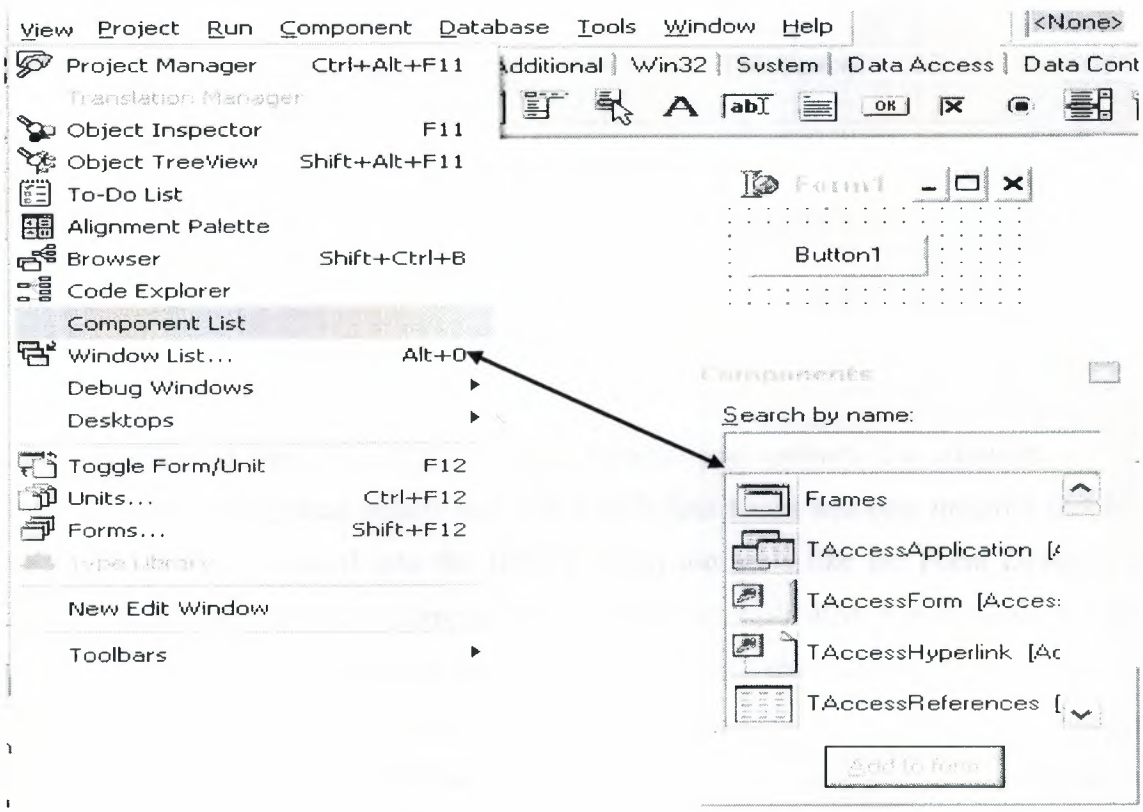


Figure. 1.28 Component List

1.4.7.a Using The Component Library

Delphi comes with a component library made up of objects, some of which are also components or controls, that you use when writing code. You can use VCL components for Windows applications and CLX components for Windows and Linux applications. The component library includes objects that are visible at Runtime such as edit controls, buttons, and other user interface elements as well-as non visual controls like datasets and timers.

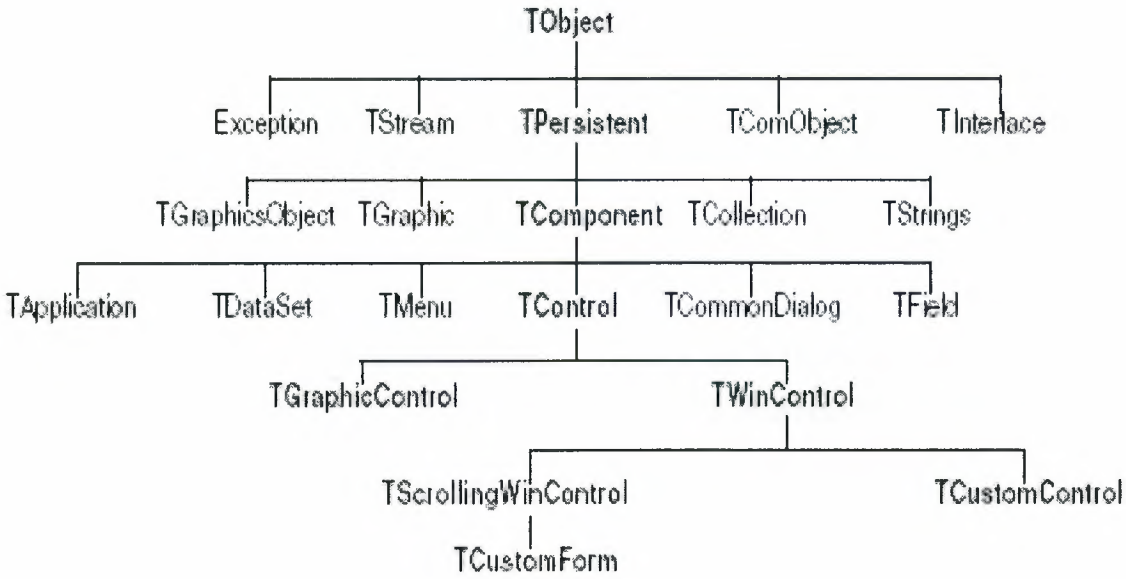


Figure. 1.29 Component Library

Objects descended from TComponent have properties and methods that allow them to be installed on the Component palette and added to Delphi forms and data modules Because the components are hooked into the IDE, you can use tools like the Form Designer to develop applications quickly. Components are highly encapsulated. For example, buttons are preprogrammed to respond to mouse clicks by firing OnClick events. If you use a button control, you don't have to write code to handle generated events when the button is clicked; you are responsible only for the application logic that executes in response to the click itself.

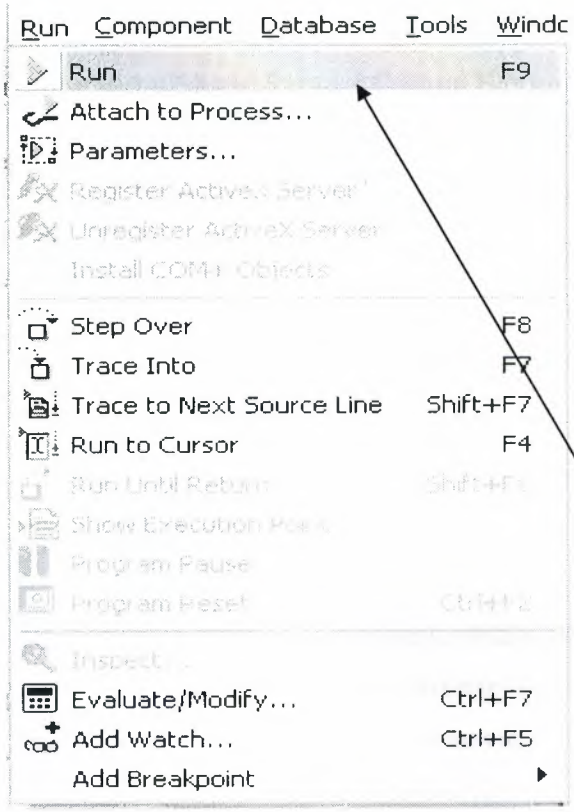
1.4.7 Compiling and Debugging Projects

After you have written your code, you will need to compile and debug your project. With Delphi, you can either compile your project first and then separately debug it, or you can compile and debug in one step using the integrated debugger. To compile your program with debug information, choose Project Options, click the Compiler page, and make sure Debug information is checked.

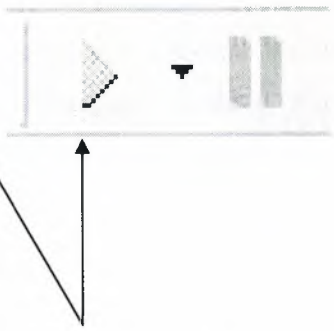
Delphi uses an integrated debugger so that you can control program execution, watch variables, and modify data values. You can step through your code line by line, examining

the state of the program at each breakpoint. To use the integrated debugger, choose Tools Debugger Options, click the General page, and make sure Integrated debugging is checked.

You can begin a debugging session in the IDE by clicking the Run button on the Debug toolbar, choosing Run Run, or pressing F9.



Choose any of the debugging commands from the Run menu. Some commands are also available on the toolbar.



RUN Button

Figure. 1.30 Compiling and Debugging

With the integrated debugger, 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. Not all debugger views are available in all editions of Delphi.

Once you set up your desktop as you like it for debugging, you can save the settings as the debugging or runtime desktop. This desktop layout will be used whenever you are debugging any application.

1.4.7.a Deploying Applications

You can make your application available for others to install and run by deploying it When you deploy an application, you will need all the required and supporting files, such as the executables, DLLs, package files, and helper applications. Delphi comes bundled with a setup toolkit called InstallShield Express that helps you create an installation program with these files. To install InstallShield Express, from the Delphi setup screen, choose InstallShield Express Custom Edition for Delphi.

1.4.7.b Internationalizing Applications

Delphi offers several features for internationalizing and localizing applications. The IDE and the VCL support input method editors (IMEs) and extended character sets to internationalize your project. Delphi includes a translation suite, not available in all editions of Delphi, for software localization and simultaneous development for different locales. With the translation suite, you can manage multiple localized versions of an application as part of a single project.

The translation suite includes three integrated tools:

- Resource DLL wizard, a DLL wizard that generates and manage resource DLLs.
- Translation Manager, a table for viewing and editing translated resources.
- Translation Repository, a shared database to store translations.

To open the Resource DLL wizard, choose File New Other and double-click the Resource DLL Wizard icon. To configure the translation tools, choose Tools| Translation Tools Options.

1.4.8 Types of Projects

All editions of Delphi support general-purpose 32-bit Windows programming, DLLs, packages, custom components, multithreading, COM (Component Object Model) and automation controllers, and multiprocess debugging. Some editions support server applications such as Web server applications, database applications, COM servers, multi-tiered applications, CORBA, and decision-support systems.

1.4.8.a Delphi (CLX Applications)

You can use Delphi, to develop cross-platform 32-bit applications that run on both the Windows and Linux operating systems. To develop a CLX application, choose File| New|CLX Application. The IDE is similar to that of a regular Delphi application, except that only the components and items you can use in a CLX application appear on the Component palette and in the Object Repository. Windows-specific features supported on Delphi will not port directly to Linux environments.

1.4.8.b Delphi (Database Applications)

Delphi offers a variety of database and connectivity tools to simplify the development of database applications. To create a database application, first design your interface on a form using the Data Controls page components. Second, add a data source to a data module using the Data Access page. Third, to connect to various database servers, add a dataset and data connection component to the data module from the previous or corresponding pages of the following connectivity tools:

• dbExpress is a collection of database drivers for cross-platform applications that provide fast access to SQL database servers, including DB2, Informix, InterBase, MSSQL, MySQL, and Oracle. With a db Express driver, you can access databases using unidirectional datasets.

• The Borland Database Engine (BDE) is a collection of drivers that support many popular database formats, including DBASE, Paradox, FoxPro, Microsoft Access, and any ODBC data source. ActiveX Data Objects (ADO) is Microsoft's high-level interface to any data source, including relational and nonrelational databases, e-mail and file systems, text and graphics, and custom business objects.

• Inter Base Express (IBX) components are based on the custom data access Delphi component architectures. IBX applications provide access to advanced Inter Base features and offer the highest performance component interface for Inter Base 5.5 and later. IBX is

compatible with Delphi's library of data-aware components. Certain database connectivity tools are not available in all editions of Delphi.

1.4.9 Administrator (BDE)

Use the BDE Administrator (BDEAdmin.exe) to configure BDE drivers and set up the aliases used by data-aware VCL controls to connect to databases.

1.4.10 Database Explorer

The SQL Explorer (DBExplor.exe) lets you browse and edit databases. You can use it to create database aliases, view schema information, execute SQL queries, and maintain data dictionaries and attribute sets.

1.4.11 Database Desktop

The Database Desktop (DBD32.exe) lets you create, view, and edit Paradox and dBase database tables in a variety of formats.

1.4.12 Data Dictionary

When you use the BDE, the Data Dictionary provides a customizable storage area, independent of your applications, where you can create extended field attribute sets that describe the content and appearance of data. The Data Dictionary can reside on a remote server to share additional information.

1.4.13 Components of custom

The components that come with Delphi are preinstalled on the Component palette and offer a range of functionality that should be sufficient for most of your development needs. You could program with Delphi for years without installing a new component, but you may sometimes want to solve special problems or display particular kinds of behavior that require custom components. Custom components promote code reuse and consistency across applications. You can either install custom components from third-party vendors or

create your own. To create a new component, choose Component New Component to display the New Component wizard.

1.4.14 Dynamic-link libraries

Dynamic-link libraries (DLLs) are compiled modules containing routines that can be called by applications and by other DLLs. A DLL contains code or resources typically used by more than one application.

1.4.15 Delphi (COM and ActiveX)

Delphi supports Microsoft's COM standard and provides wizards for creating ActiveX controls. Choose File New Other and click the ActiveX tab to access the wizards. Sample ActiveX controls are installed on the ActiveX page of the Component palette. Numerous COM server components are provided on the Servers tab of the Component palette. You can use these components as if they were VCL components. For example, you can place one of the Microsoft Word components onto a form to bring up an instance of Microsoft Word within an application interface.

1.4.16 Component Type Libraries

Type libraries are files that include information about data types, interfaces, member functions, and object classes exposed by an ActiveX control or server. By including a type library with your COM application or ActiveX library, you make information about these entities available to other applications and programming tools. Delphi provides a Type Library editor for creating and maintaining type libraries.

1.5 Work Area (IDE)

The IDE provides many tools to support development, so you'll want to reorganize your work area for maximum convenience. You can rearrange menus and toolbars, combine tool windows, and save your new desktop layout.

1.5.1 Arranging Menus and Toolbars

In the main window, you can reorganize the menu, toolbars, and Component palette by clicking the grabber on the left-hand side of each one and dragging it to another location.

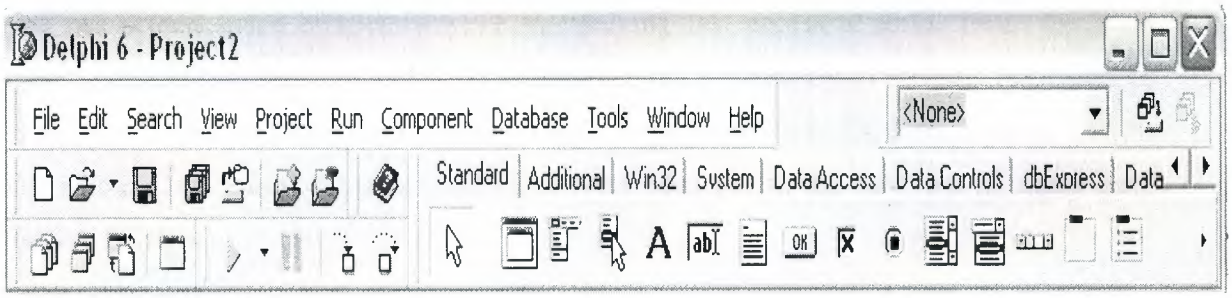


Figure. 1.31 Arranging Menus and Toolbars

You can separate parts from the main window and place them elsewhere on the screen or remove them from the desktop altogether. This is useful if you have a dual monitor setup.

You can add or delete tools from the toolbars by choosing View Toolbars Customize. Click the Commands page, select a category, select a command, and drag it to the toolbar where you want to place it.

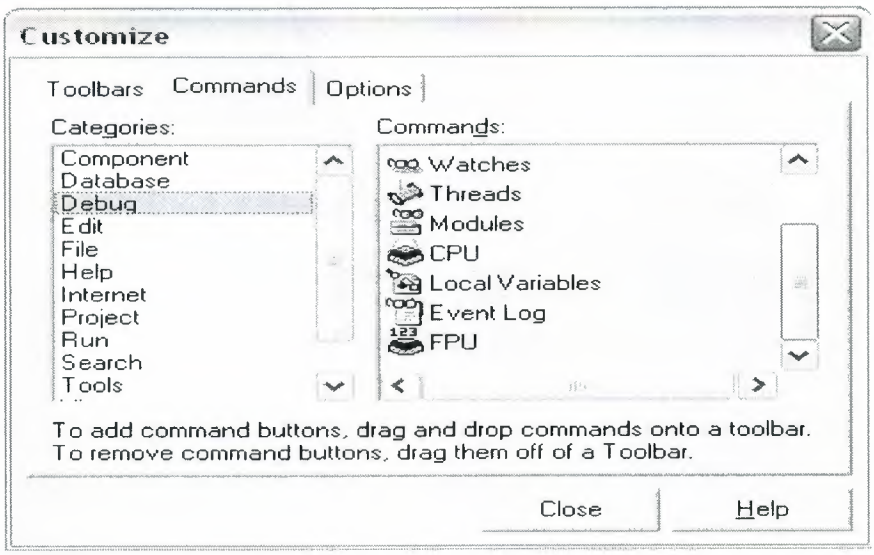


Figure. 1.32 Customize Command

1.5.2 Tool Windows

You can open and close individual tool windows and arrange them on the desktop as you wish. Many windows can also be docked to one another for easy management. Docking which means attaching windows to each other so that they move Together helps you use screen space efficiently while maintaining fast access to tools. From the View menu, you can bring up any tool window and then dock it directly to another. For example, when you first open Delphi in its default configuration, the Code Explorer is docked to the left of the Code editor. You can add the Project Manager to the first two to create three docked windows.

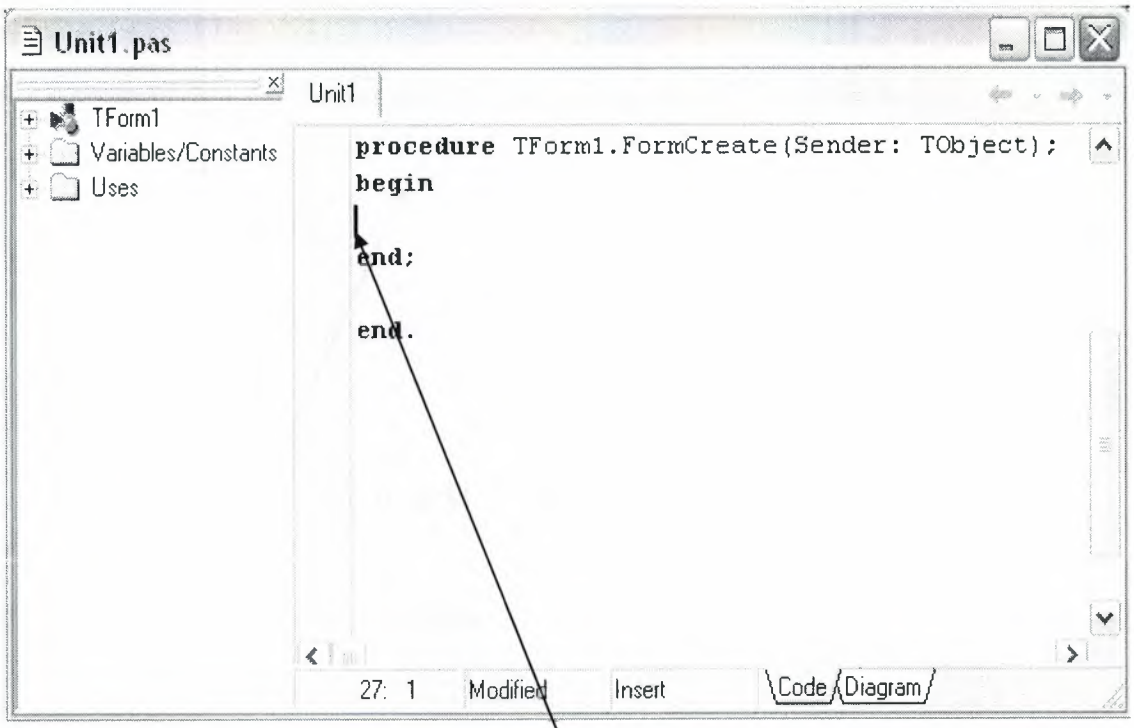


Figure. 1.33 Docking Tool Windows

Here the Project Manager and Code Explorer are docked to the Code editor. You can combine, or "dock" windows with either grabbers, as on the right, or tabs, as on page 5-4.

To dock a window, click its title bar and drag it over the other window. When the drag outline narrows into a rectangle and it snaps into a corner, release the mouse. The two windows snap together.

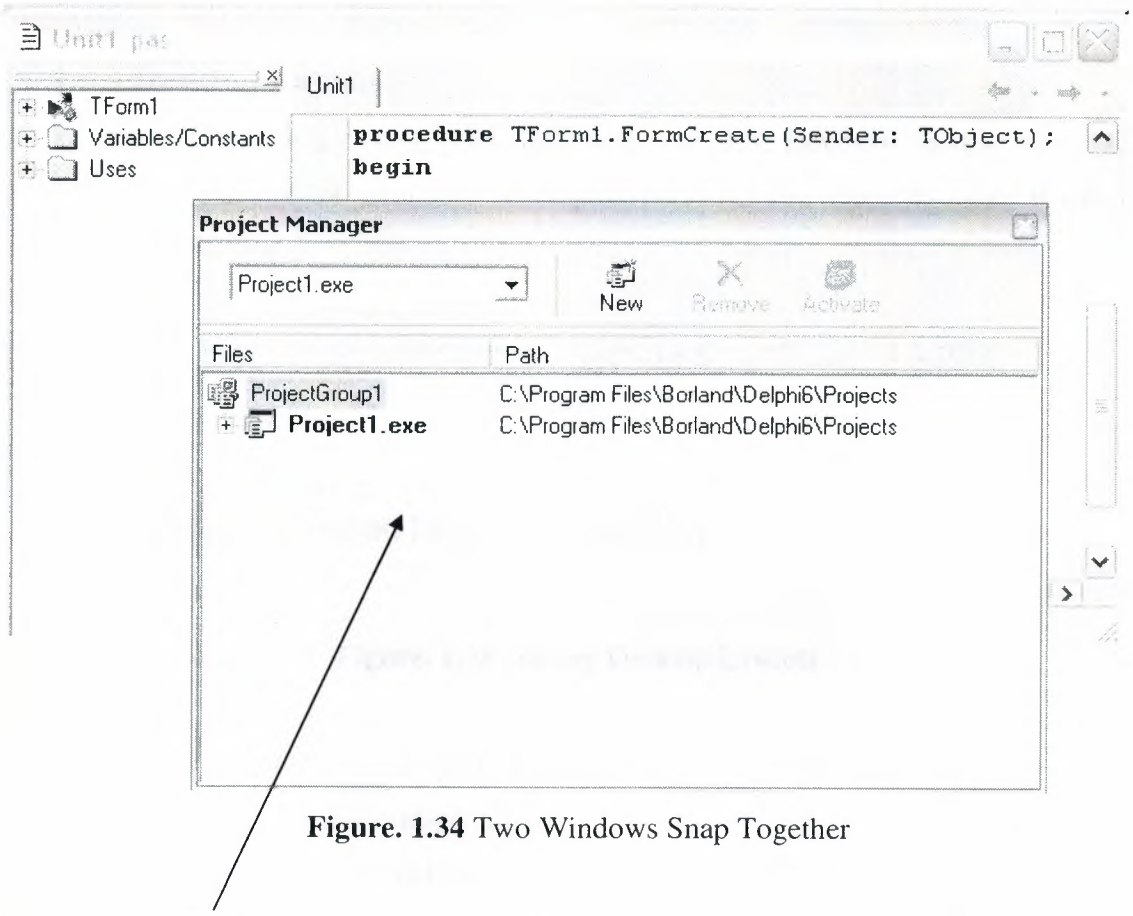


Figure. 1.34 Two Windows Snap Together

To get docked windows with grabbers, release the mouse when the drag outline snaps to the window's corner.

You can also dock tools to form tabbed windows.

To undock a window, double click its grabber or tab, or click and drag the tab outside of the docking area. To turn off automatic docking, either press the Ctrl key while moving windows around the screen, or choose Tools|Environment Options, click the references page, and uncheck the Auto drag docking check box.

1.5.3 Desktop Layouts

You can customize and save your desktop layout. The Desktops toolbar in the IDE includes a pick list of the available desktop layouts and two icons to make it easy to customize the desktop.

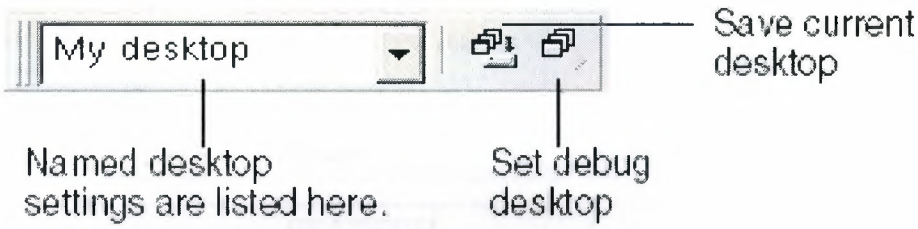


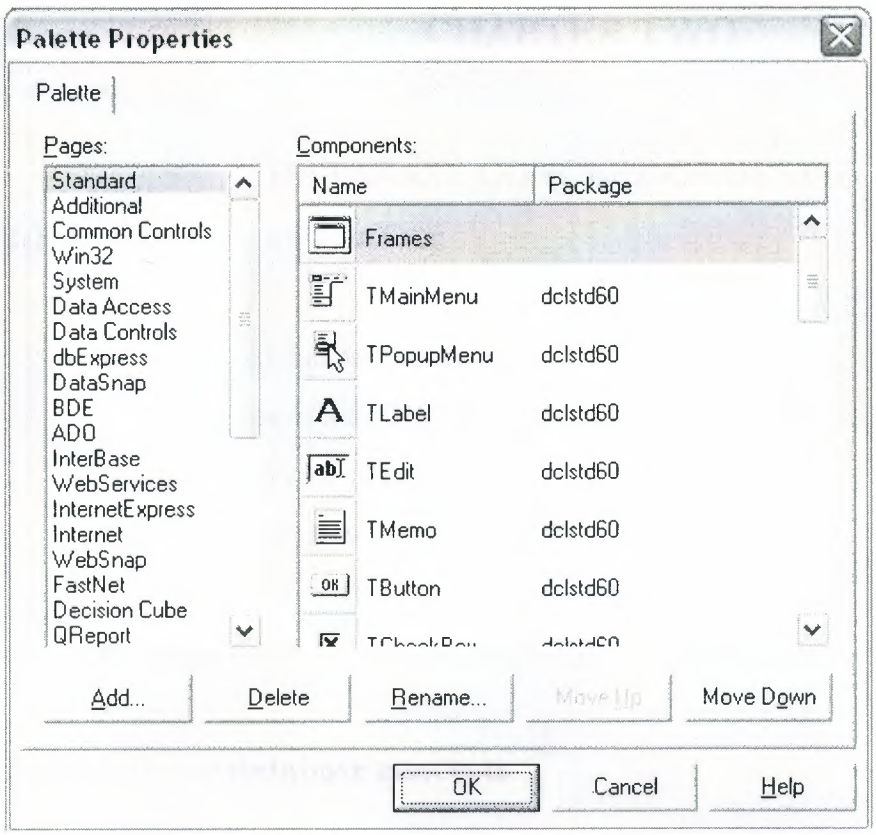
Figure. 1.35 Saving Desktop Layouts

Arrange the desktop as you want, including displaying, sizing, and docking particular windows. On the Desktops toolbar, click the Save current desktop icon or choose View Desktops Save Desktop, and enter a name for your new layout.

1.6 The Component Palette

To add, delete, rearrange, or rename pages, or to hide or rearrange components, use the Palette Properties dialog box. You can open this dialog box in several ways:

- Choose Component Configure Palette.
- Choose Tools Environment Options and click the Palette tab.
- Right-click the Component palette and choose Properties.



You can rearrange the palette and add new pages.

Figure. 1.36 Palette Properties Dialog Box

1.6.1 Creating Component Templates

Component templates are groups of components that you add to a form in a single operation. Templates allow you to configure components on one form, then save their arrangement, default properties, and event handlers on the Component palette to reuse on other forms.

To create a component template, simply arrange one or more components on a form and set their properties in the Object Inspector, and select all of the components by dragging the mouse over them. Then choose Component|Create Component Template. When the Component Template Information dialog box opens, select a name for the template, the palette page on which you want it to appear, and an icon to represent the template on the palette.

CHAPTER TWO

DATABASE CONCEPT OF DELPHI 6

2.1 Architecture of database

- Relational database concepts
- The pieces of a database system
- How the pieces fit together
- Multi-tier computing architecture
- Using multiple databases
- About dbase
-

2.2 Relational database concepts

A relational database-management system (RDBMS) is a system for storing and retrieving data, in which the data is organized into interrelated tables.

SQL Anywhere Studio provides two relational database systems. Adaptive Server Anywhere is the primary, full featured RDBMS, with a multitude of uses, from a network database server hosting many clients to a compact embedded database. UltraLite is a small-footprint relational database. The UltraLite deployment technology allows you to use Adaptive Server Anywhere features on even the smallest of devices.

- Database tables
- Relations between tables
- Other database objects

2.3 Accessing data in other databases

You can access databases on multiple database servers, or even on the same server, using the Adaptive Server Anywhere Remote Data Access features. The application is still connected to a single database as in the architecture diagrams above, but by defining remote servers, you can use proxy tables that exist on the remote database as if they were in the database to which you are connected.

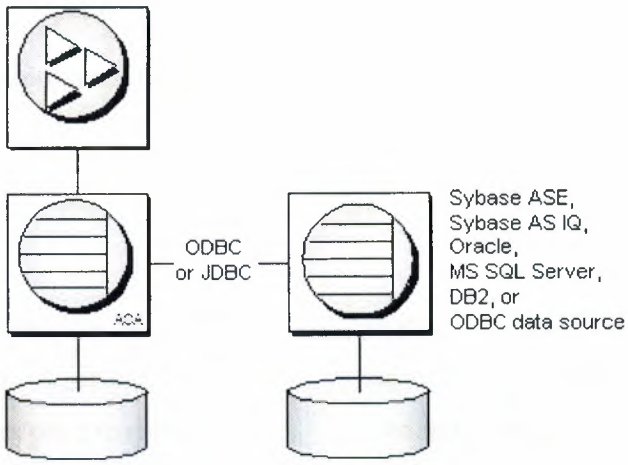


Fig. 2.1 Relation Diagram

2.4 DBASE IV Table Specification

The dBASE IV table format was introduced in dBASE IV for DOS. Following are the specifications for dBASE IV tables:

- 2GB file size.
- Two billion records per file.
- A maximum of 255 fields per record.

Maintained indexes can have up to 47 indexes per file. Each index can be created using field expressions of virtually any combination, including conditional expressions of up to 255 characters per expression that result in an index of up to 100 bytes. Unlimited nonmaintained indexes can be stored on disk. You can use up to 47 of them simultaneously.

2.5 dBase V Table Specifications

The dBASE V table format was introduced in dBASE V for Windows. Following are the specifications for dBASEA V tables.

- Up to one billion records per file.
- A maximum of 1,024 fields per record.
- Up to 32,767 bytes per record.
- Unlimited nonmaintained indexes can be stored on disk. You can use up to 47 of them simultaneously.
- Up to 10 master index files open per database. Each master index can have up to 47 indexes.
- Maintained indexes can have up to 47 indexes per file. Each index can be created using field expressions of virtually any combination, including conditional expressions of up to 255 characters per expression that result in an index of up to 100 bytes.

2.6 DBASE Field Types

Character (C)

DBASE III+, IV, and V field type that can contain up to 254 characters (including blank spaces). This field is similar to the Paradox Alpha field type.

Date (D)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V. dBASE tables can store dates from January 1, 100, to December 31, 9999. Paradox 5 tables can store from 12/31/9999 B.C. to 12/31/9999 A.D.

Float (F)

DBASE IV, and V floating-point numeric field type provides up to 20 significant digits.

Logical (L)

Paradox 5 and 7 and dBASE III+, IV, and V field type can store values representing True or False (yes or no). By default, valid entries include T and F (case is not important).

Memo (M)

Paradox 4, 5, and 7 as well as dBASE III+, IV, and V field. A Paradox field type is an Alpha variable-length field up to 256MB per field. dBASE Memo fields can contain binary as well as memo data.

OLE (O)

Paradox 1, 5, and 7 as well as dBASE V field type that can store OLE data.

Number (N)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V field type can store up to 15 significant digits -10307 to + 10308 with up to 15 significant digits.

DBASE number fields contain numeric data in a Binary Coded Decimal (BCD) format. Use number fields when you need to perform precise calculations on the field data. Calculations on number fields are performed more slowly but with greater precision than are calculations on float number fields. The size of a dBASE number field can be from 1 to 20. Remember, however, that BCD is in Paradox 5 and 7 only for compatibility and is mapped directly to the Number field type.

Short (S)

Paradox 3.5, 4, 5, and 7 field type that can contain integers from -- 32,767 through 32,767 (no decimal)

2.7 Paradox Standard Table Specifications-

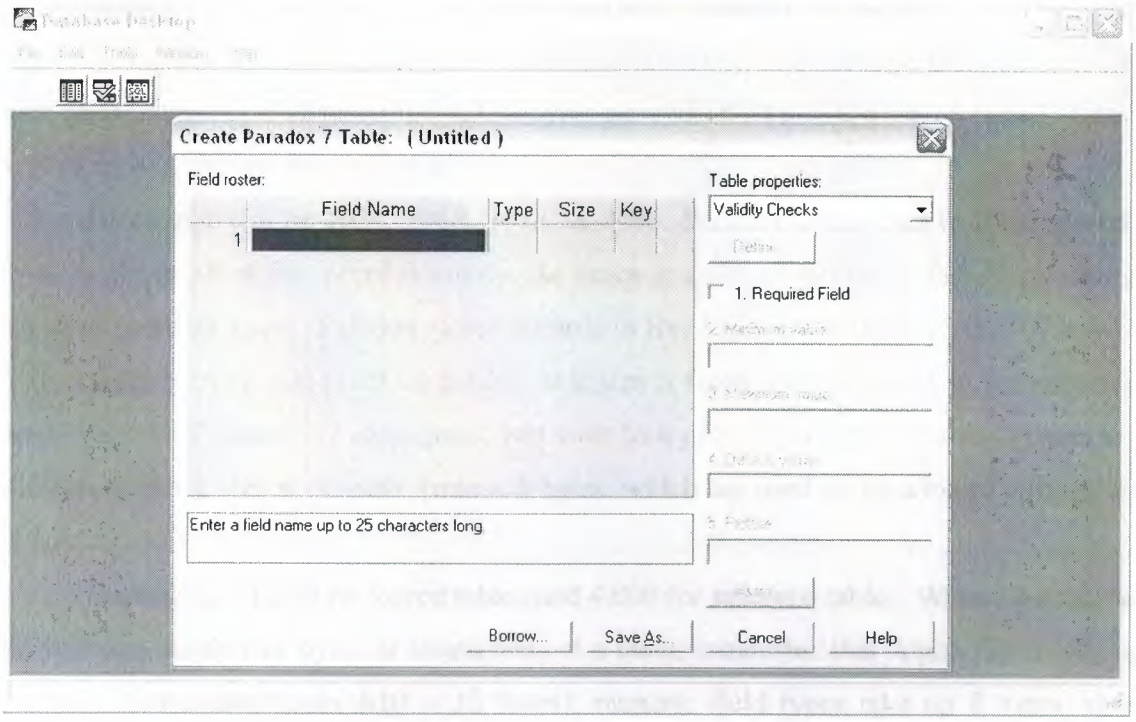


Fig. 2.2 Paradox Standart Table

2.7.1 Paradox 4 table structure.

The Paradox standard table format was introduced in Paradox for DOS version 4. Other products that use the standard format include Paradox for DOS version 4.5, ObjectVision 2.1, and Paradox for Windows versions 1.0 and 4.5.

Earlier versions of the Paradox table type are referred to as the Compatible table type. In the BDE Configuration Utility, the level option for the Paradox driver dictates what default table type is created by Paradox for Windows. Use 3 for Compatible tables, 4 for Standard tables (the default). Following are the specifications for standard Paradox tables:

- 256MB file size limit if the table is in Paradox format and using a 4K block size.
- Up to 255 fields per record.
- Up to 64 validity checks per table.
- A primary index can have up to 16 fields.

- Tables can have up to 127 secondary indexes.
- Up to two billion records per file.

Because of the 256MB file size limit and other factors such as block size, however, the limit is much smaller. Tables of 190,000 records are easily achievable (and you can have more if you don't use up the 1,350-bytes-per-record limit for a keyed table). Tables with close to a million records are common.

Block size can be 1024, 2048, 3072, or 4096. Paradox stores data in fixed records. Even if part or all of the record is empty, the space is claimed. Knowing the interworkings can save you disk space. Paradox stores records in fixed blocks of 1024, 2048, 3072, 4096 in size. After a block size is set for a table, that size is fixed, and all blocks in the table will be of that size. To conserve disk space, you want to try to get your record size as close to a multiple of block size as possible (minus 6 bytes, which are used by Paradox to manage the table).

Record size. 1,350 for keyed tables and 4,000 for unkeyed tables. When figuring out the size (the number of bytes or characters) of a table, remember that Alpha fields take up their size (for example, an A10 = 10 bytes), numeric field types take up 8 bytes, short number field types take up 2 bytes, money takes up 8, and dates take up 4 bytes.
Memos, BLOBs, and so on take 10 bytes plus however much of the memo is stored in the .DB. For example, M15 takes 25 bytes.

2.7.2 Paradox 5 Table Specifications

The Paradox 5 table format was introduced in Paradox for Windows version 5.
Following are the specifications for Paradox 5 tables:

- Up to two billion records per file.
- File size is limited to two gigabytes.
- Up to 255 fields per record.

Record size: Up to 10,800 bytes per record for indexed tables and 32,750 bytes per record for nonindexed tables. When figuring out the size (the number of bytes or characters) of a

table, remember that Alpha fields take up their size (for example, an A10 = 10 bytes), numeric field types take up 8 bytes, short number field types take up 2 bytes, money takes up 8, and dates take up 4 bytes.

Memos, BLOBs, and so on take 10 bytes plus however much of the memo is stored in the .DB. For example, M15 takes 25 bytes.

Up to 64 validity checks per table for Paradox for Windows tables.

A primary index can have up to 16 fields.

Tables can have up to 127 secondary indexes.

Block size can be from 1K to 32K in steps of 1K. For example, 1024, 2048, 3072, 4096, 5120...32768.

2.7.3 Paradox 7 and Above Table Specifications

The Paradox 7 table format was introduced in Paradox version 7 for Windows 95/NT. The Paradox 7 table format has all the same specifications as the Paradox 5 table format with two additions. Following are the specification additions for the Paradox 7 table format:

- Added descending secondary indexes.
- Added unique secondary indexes

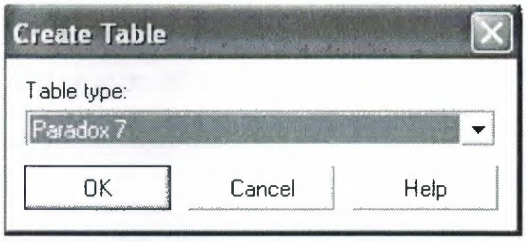


Fig. 2.3 Paradox Create Table

2.7.4 Paradox Field Types

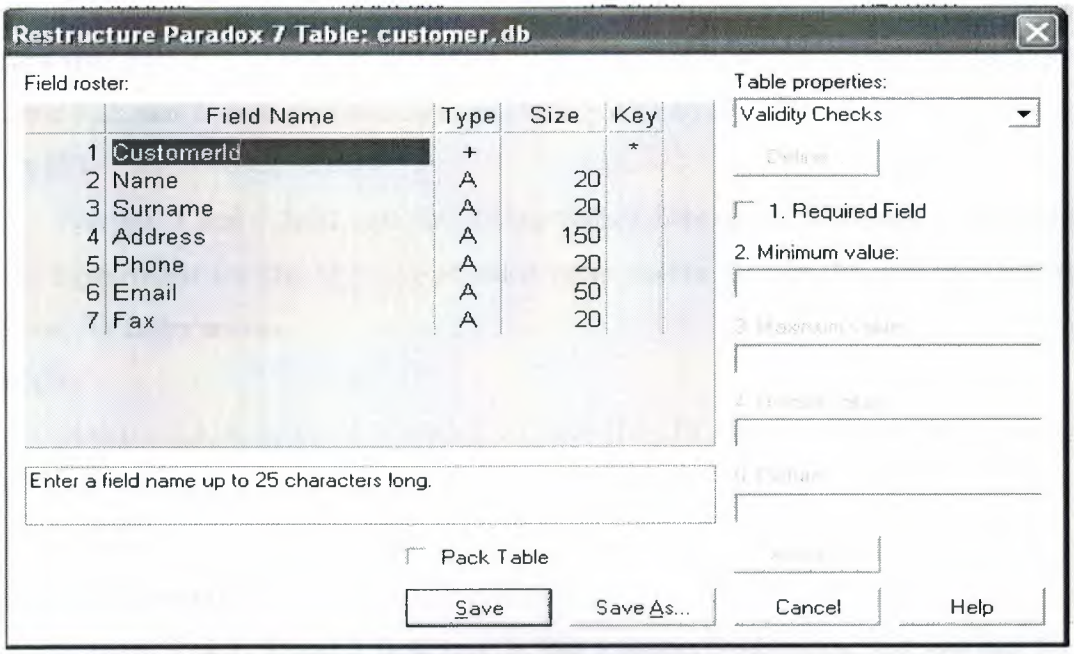


Fig. 2.4 Paradox fields type

Alpha (A)

Paradox 3.5, 4, 5, and 7 field type that can contain up to 255 letters and numbers. This field type was called Alphanumeric in versions of Paradox before version 5. It is similar to the Character field type in dBASE.

Autoincrement (+)

Field type introduced in the Paradox 5 table format that adds one to the highest number

in the table whenever a record is inserted. The starting range can from -2,147,483,647 to 2,147,483,647. Deleting a record does not change the field values of other records.

BCD (#)

Paradox 5 and 7 field type which is provided only for compatibility with other applications that use BCD data. Paradox correctly interprets BCD data from other

applications that use the BCD type. When Paradox performs calculations on BCD data, it converts the data to the numeric float type, then converts the result back to BCD. When this field type is fully supported, it will support up to 32 significant digits.

Binary (B)

Paradox 1, 5, and 7 field type that can store binary data up to 256MB per field.

Bytes (Y)

Paradox 5 and 7 field type for storing binary data up to 255 bytes. Unlike binary fields, bytes fields are stored in the Paradox table (rather than in the separate .MB file), allowing for faster access.

Date (D)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V. dBASE tables can store dates from January 1, 100, to December 31, 9999. Paradox 5 tables can store from 12/31/9999 B.C. to 12/31/9999 A.D.

Formatted Memo (F)

Paradox 1, 4.5, 5, and 7 field type is like a memo field except that you can format the text. You can alter and store the text attributes of typeface, style, color, and size. This rich text document has a variable-length up to 256MB per field.

Graphic (G)

Paradox 1, 5, and 7 field type can contain pictures in .BMP (up to 24 bit), .TIF (up to 256 color), .GIF (up to 256 color), .PCX, and .EPS file formats. Not all graphic variations are available. For example, currently you cannot store a 24-bit .TIF graphic. When you paste a graphic into a graphic field, Paradox converts the graphic into the .BMP format.

Logical (L)

Paradox 5 and 7 and dBASE III+, IV, and V field type can store values representing True or False (yes or no). By default, valid entries include T and F (case is not important).

Memo (M)

Paradox 4, 5, and 7 as well as dBASE III+, IV, and V field. A Paradox field type is an Alpha variable-length field up to 256MB per field. dBASE Memo fields can contain binary as well as memo data.

For Paradox tables, the file is divided into blocks of 512 characters. Each block is referenced by a sequential number, beginning at zero. Block 0 begins with a 4-byte number in hexadecimal format, in which the least significant byte comes first. This number specifies the number of the next available block. It is, in effect, a pointer to the end of the memo file. The remainder of Block 0 isn't used.

Money ($)

Paradox 3.5, 4, 5, and 7 field type, like number fields, can contain only numbers. They can hold positive or negative values. Paradox recognizes up to six decimal places when performing internal calculations on money fields. This field type was called Currency in previous versions of Paradox.

OLE (O)

Paradox 1, 5, and 7 as well as dBASE V field type that can store OLE data.

Number (N)

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V field type can store up to 15 significant digits -10307 to + 10308 with up to 15 significant digits.

dBASE number fields contain numeric data in a Binary Coded Decimal (BCD) format. Use number fields when you need to perform precise calculations on the field data. Calculations on number fields are performed more slowly but with greater precision than are calculations on float number fields. The size of a dBASE number field can be from 1 to 20. Remember, however, that BCD is in Paradox 5 and 7 only for compatibility and is mapped directly to the Number field type.

Short (S)

Paradox 3.5, 4, 5, and 7 field type that can contain integers from -- 32,767 through 32,767 (no decimal).

Time (T)

Paradox 5 and 7 field type that can contain time times of day, stored in milliseconds since midnight and limited to 24 hours.This field type does not store duration which is the difference between two times. For example, if you need to store the duration of a song, use an Alpha field. Whenever you need to store time, make a distinction between clock time and duration. The Time field type is perfect for clock time. Duration can be stored in an Alpha field and manipulated with code.

TimeStamp (@)

Paradox 5 field type comprised of both date and time values. Rules for this field type are the same as those for date fields and time fields.

CHAPTER THREE

USER MANUEL

3.1 User login and license register

This part is user's entry screan.They join the program with entering password and user name which is given by admin.

Figure3.1.1

If the program setup firstly username is admin and the password is 987832525.With entering these infornations you will see this error.

Figure3.1.2

The error in **Figure 3.1.2** is because setup is using in its first time.This shows doctor's and user's haven't any registeration. This form takes places when you push OK button.

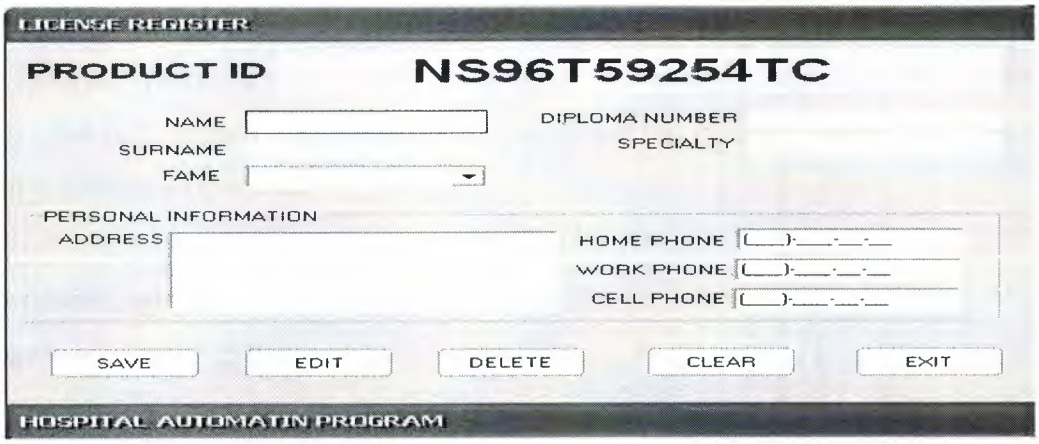


Figure3.1.3

When this part is filling the informations must enter carefully.Because this informations is just enter for one time.And this informations directly take place in reports and recipe.After filling the form you must become the owner of licence with joining serial register also this is for onyl one.This caused you must be careful.

Figure3.1.4

The serial informations will automaticly send to seller.The product Id is so important that will use in every problem and process.After entering the correct values the message which is above will taken.

Figure3.1.5

After this the program completely will automatictly make licence's register.By entering incorrect values of registiration the program will automaticly close itself.When completing the registiration of licence the program must run again by using correct user name and password to reach the main menu.

First username: admin

First password: 987832525

Firstly User must define new user when user reach the main menu. You can use the program with the first username and first password but we dont definetely advice this.

3.2 User register

New user must reach the setting option from the main menu,then new user register form.In this part you can define new user how much you want and you can let the users to manage the program.

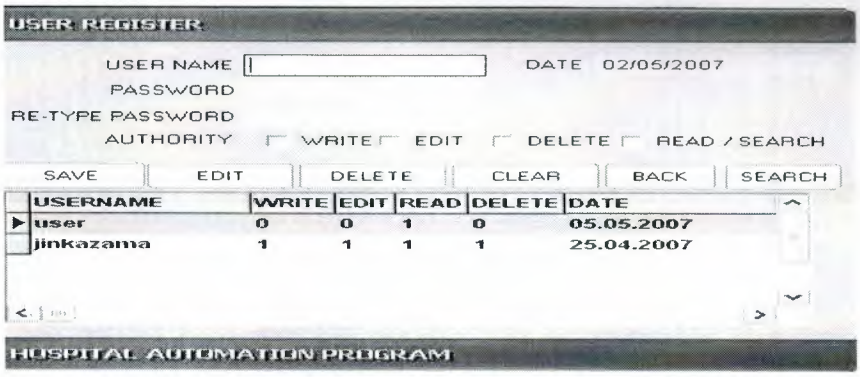


Figure3.2.1

This part only uses for whole authorized.For other users it give a error message.

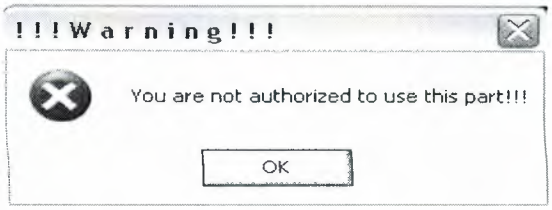


Figure3.2.2

The user who you defined must join from the beginning.In **Figure3.1.1** you can register or you can use setting option then user login form.

3.3 File register

3.3.1 New file register

This form is use patients who come at first time.A file opens which you can register all informations about patients.This operation can be done only one time.There will not be the same file number or the same name which recorded before.

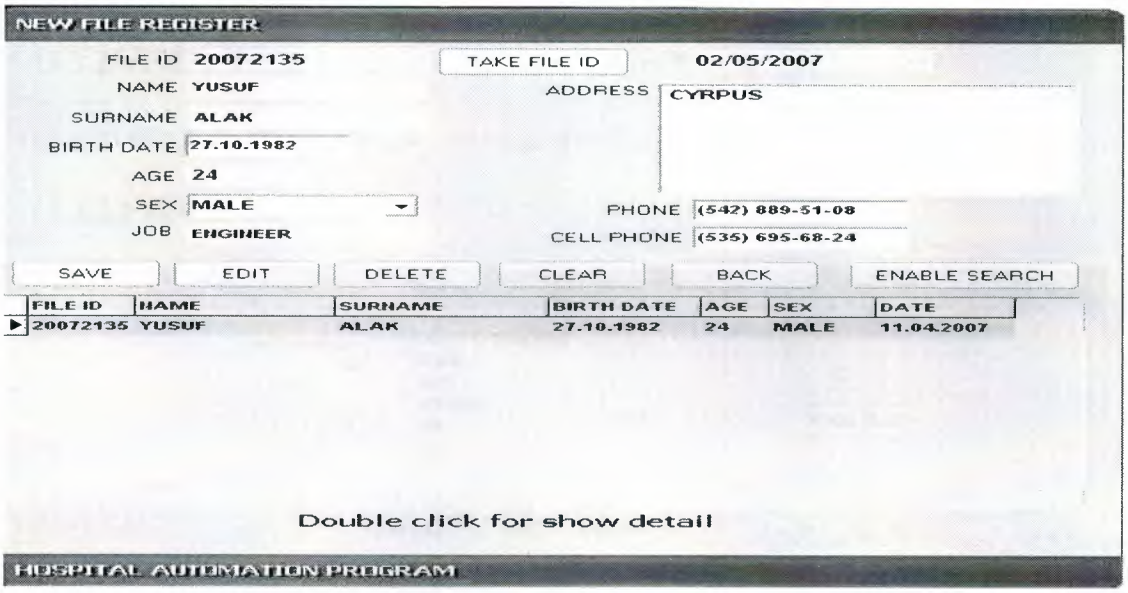


Figure3.3.1.1

For one patient only one file number can be given.The first number of file number will be numbers of the year when patient register.Last four numbers will be chosen randomly.And the program will never give the same file number to another patient.For every year the capasity of file register is 9999.

If you insistently want to register a patient although all warnings,you get the message below.

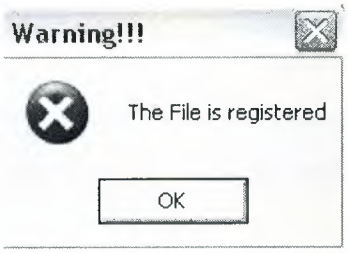


Figure3.3.1.2

After that you can not register.

3.3.2 File delete, search and edit

You can reach a patient's informations more than one way.

3.3.2.a FILE ➔ LIST (This part can be used)
3.3.2.b FILE ➔ NEW FILE bölümünden (Also this part can be used)

After you reach an information with using one of that way,you can delete or edit.

3.3.2.a File ➔ list part

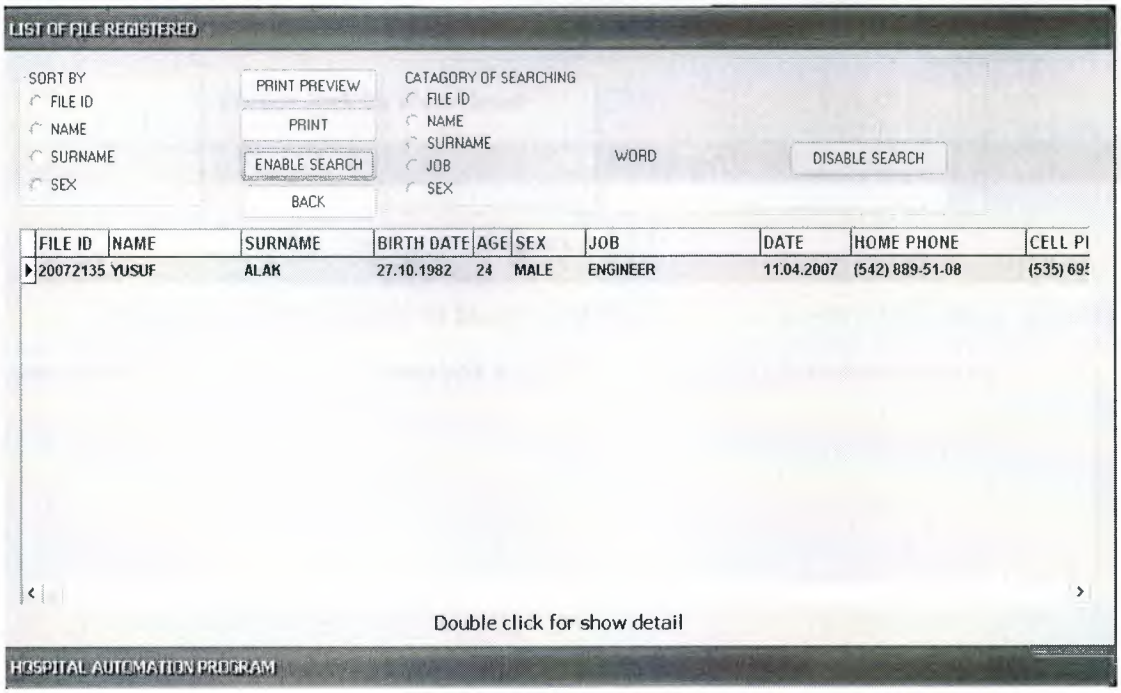


Figure3.3.2.a

From this part you can search and with double clicking,you can reach more detail information about your record.

3.3.2.b File ➔ New file Part

If you want to search in part which is above,you need to click Enable Search button.By the openning window you can reach easily the person who you want with entering the informations.

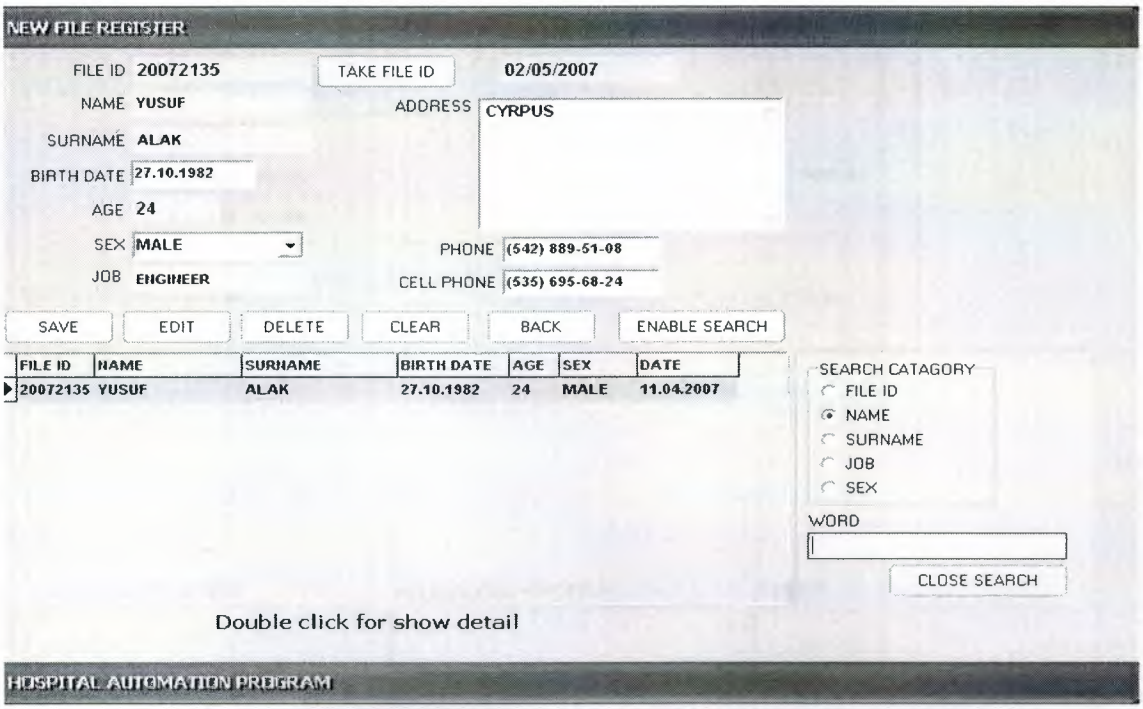


Figure3.3.2.b

When a patient comes to hospital at the first time patient will get a file id and the patient must know file id and tell it to the authorized by the next time.

3.4 Patient register

3.4.1 New patient register

This part includes a patient's informations which is recorded before with the patient's first coming.Also you can store every type of inspection and medicines which are given by the doctor.

PATIENT REGISTER

FILE ID 20072135 REGISTER DATE 02/05/2007 MORE INFORMATION

NAME YUSUF TENSION ___,___ MM HG

SURNAME ALAK TEMRATURE __ C

MEDICAL DIAGNOSIS PULSE __ M

RESPIRATION SYSTEM MEDICAL DIAGNOSIS SEARCH

KVS

GIS

HEAD & NECK EKSTREMITE VD. UROLOGY SYSTEM

COMMON SITUATION NEUROLOGY SYSTEM OTHER

FILE ID	NAME	SURNAME	DATE	MEDICAL DIAGNOSIS
20072135	YUSUF	ALAK	11.04.2007	FLUE

SAVE EDIT DELETE CLEAR FORM BACK

Figure3.4.1.a

If a patient got a file id before and with the chosen of file id,you can see the last all information about the patient.Also if you want you can edit or create a new register

or you can delete all the information about a patient.If you want to laboratory results and applied cure or given medicines, you need to click More Information button.

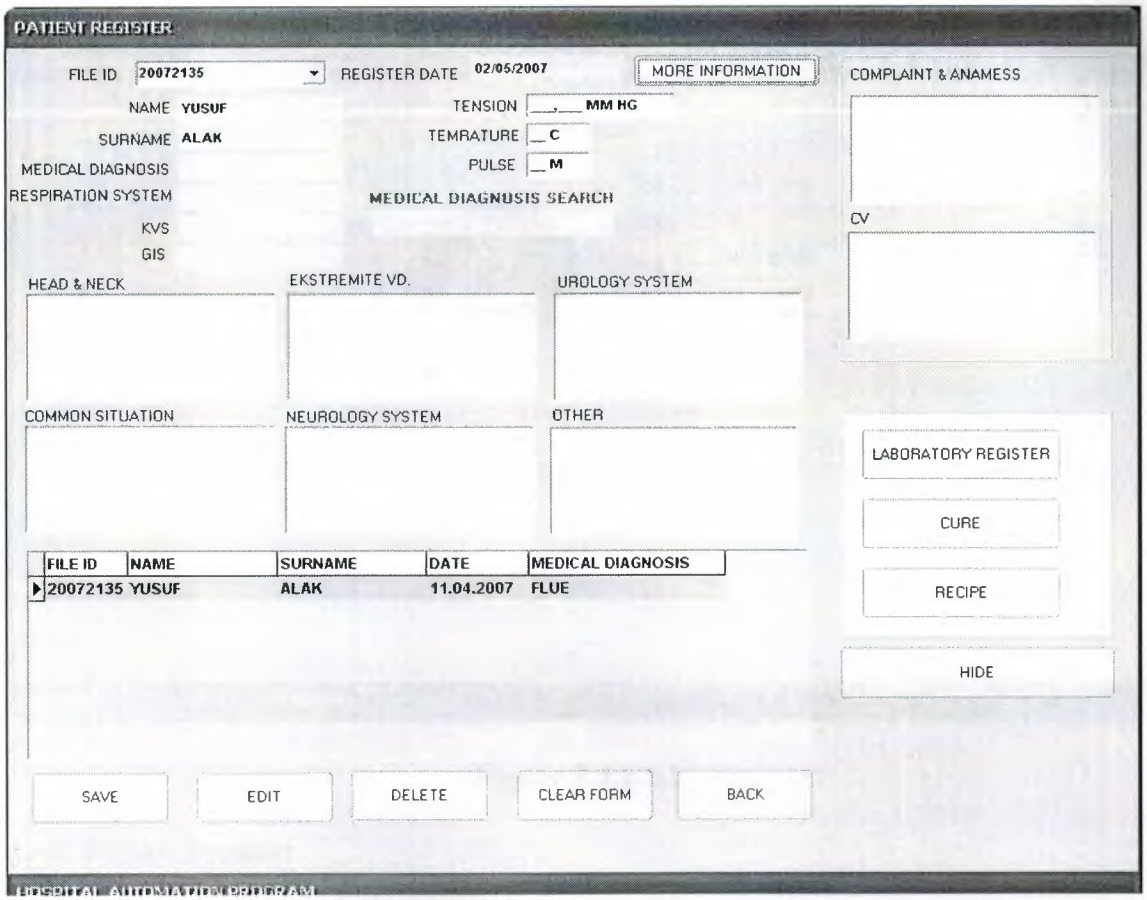


Figure3.4.1.b

By the clicking Laboratory register button:

3.4.1.1 laboratory register

You can register patient's laboratory results also you can reach last informations.

LABORATORY REGISTER OF SICK

FILE ID 20072135 DATE 02/05/2007

NAME YUSUF SURNAME ALAK

Aks T.KD

Tirigliserit DL - KD KOL

BUN

Crea

ALT

AST

PLT

Hb (GR : DL)

Htc %

Lokosit ML : CUBE

Guton ML : CUBE

Sedim MM : H

LKG

TELE_PA Lungs

Urine

SAVE EDIT DELETE CLEAR PRINT PRIVIEW PRINT BACK

FILE ID	NAME	SURNAME	DATE	AKS	TIRIGLISERIT	BUN	CREA	ALT
20072135	YUSUF	ALAK	11.04.2007	456 T.kd	566 DI -	5646	46545	564

HOSPITAL AUTOMATIN PROGRAM

Figure 3.4.1.1.1

3.4.1.2 Cure register

If you need to reach a patient's informations, medical diagnosis or applied cure this Cure Register will help you.

SICK CURE REGISTER

FILE ID 20072135 DATE 02/05/2007

NAME YUSUF MEDICAL DIAGNOSIS

SURNAME ALAK CHECK DATE 19/03/2007

FRONT CURE CURE TO APPROVE

SAVE EDIT DELETE CLEAR BACK SEARCH

FILEID	NAME	SURNAME	DATE	MEDICAL DIAGNOS
20072135	YUSUF	ALAK	23.04.2007	FLUE

HOSPITAL AUTOMATION PROGRAM

Figure3.4.1.2

3.4.1.3 Recipe register

In part of Recipe Register you can give new medicines or reach to the last medicines which are given to the patient.

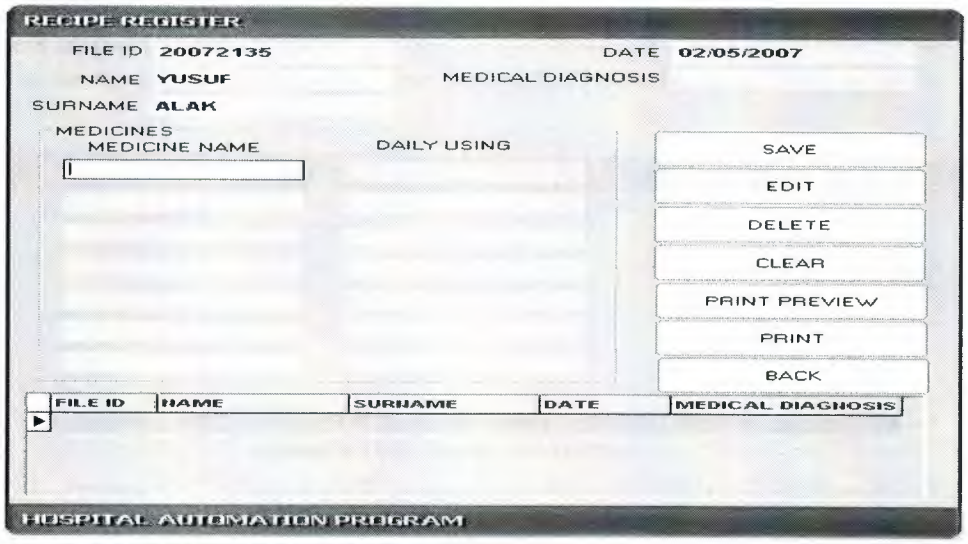


Figure3.4.1.3

3.4.2 List of patient register

In this part you can reach a patient's all informations with double clicking the wanted information.

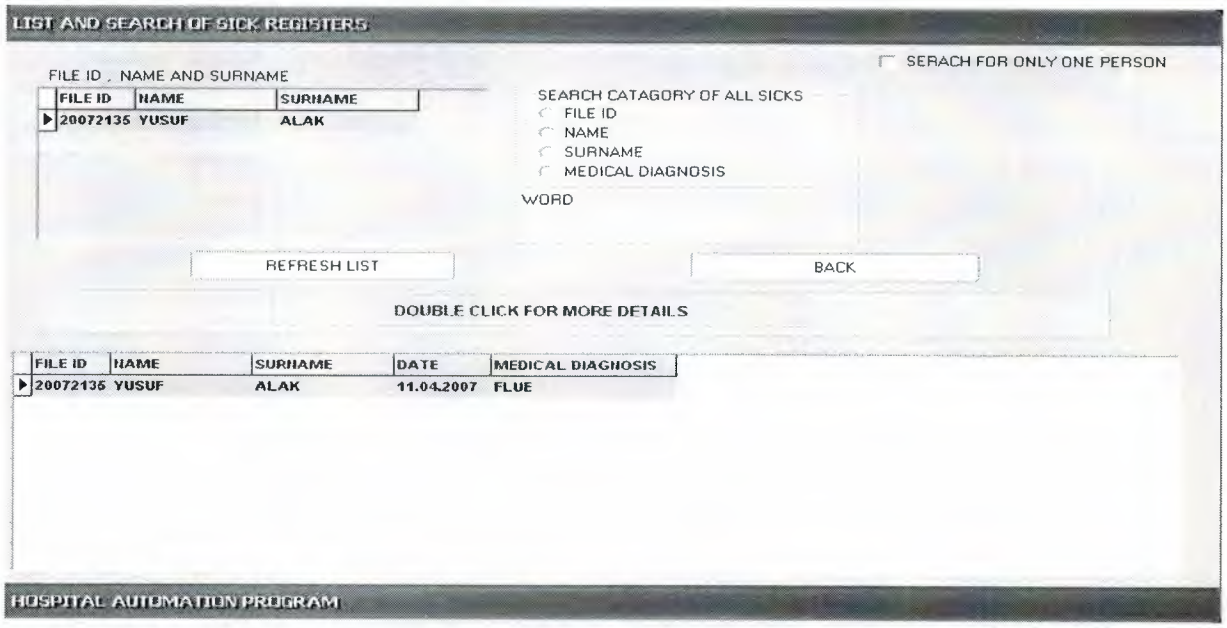


Figure3.4.2

You can search catogory of all patients.By double clicking you can have more details. In Figure 3.4.2.

3.5 Appointment register

3.5.1 New appointment register

In the part of Appointment Register you can register appointment of patients.

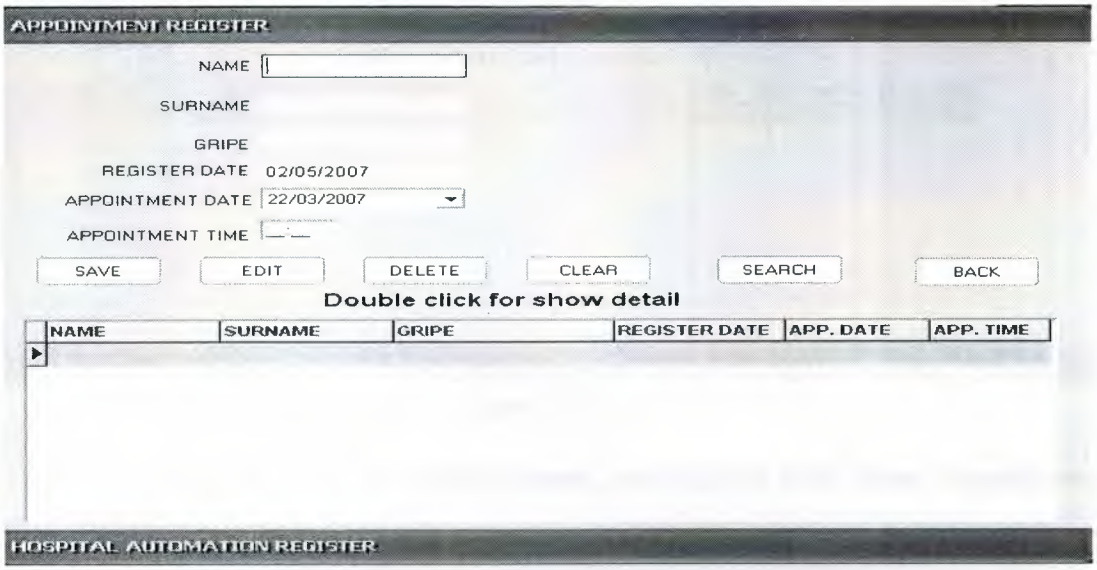


Figure3.5.1.a

You can search in catagory of name by the clicking of search button.

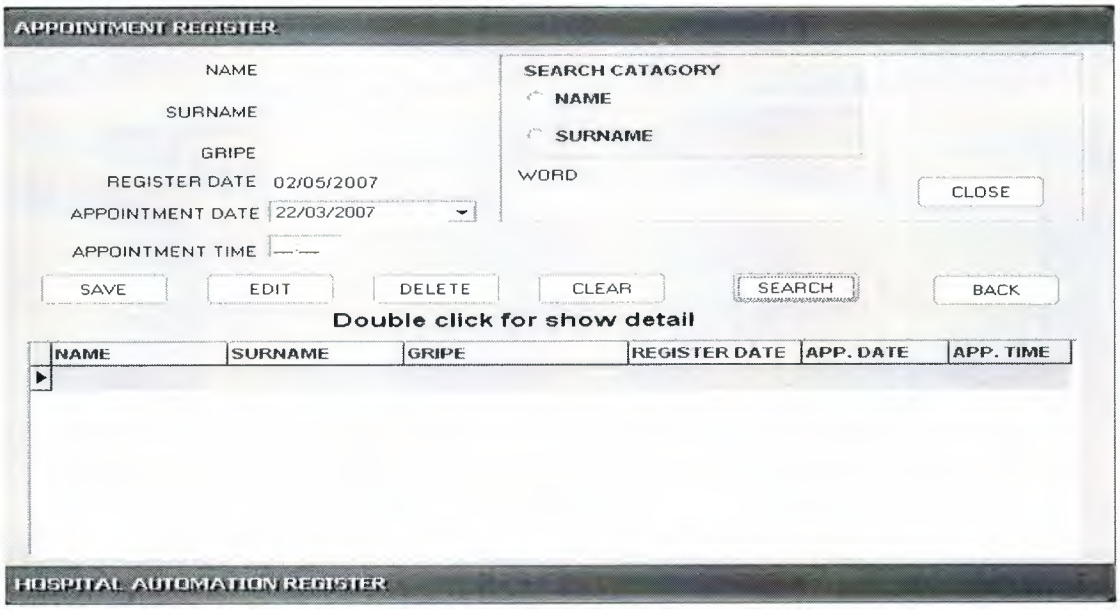


Figure3.5.1.b

3.5.2 List of appointment register

You can order by the patients in which catagory you want.By double clicking you can reach more information.

Figure3.5.2.a

If you want to reach daily appointment, you need to click **Show Appointment** which is in the **Main Menu**.

Figure3.5.2.b

CHAPTER FOUR

PROGRAM SOURCE CODE

4.1 Mainmenu codes

4.1.1 Necessary functions

```
var
 Form1: TForm1;
        a,b:integer;
        a1:string;
w,e,r,d:integer;
procedure read;
begin
form9.tfXPEdit1.Text:=form1.Query1.Fields[0].AsString;
form9.tfXPEdit2.Text:=form1.Query1.Fields[1].AsString;
form9.tfXPEdit3.Text:=form1.Query1.Fields[2].AsString;
form9.tfXPEdit5.Text:=form1.Query1.Fields[3].AsString;
form9.DateTimePicker1.Date:=strtodate(form1.Query1.Fields[4].AsString);
form9.maskEdit1.Text:=form1.Query1.Fields[5].AsString;
end;
function srcd(a:string;b:string):boolean;
begin
srcd:=false;
form1.Query1.First;
while not form1.Query1.eof do
if (a=form1.Query1.Fields[0].asstring) and (b=form1.Query1.Fields[1].asstring)then
begin
srcd:=true;
exit;
end
else
form1.Query1.Next;
end;
```

4.1.2 Form create

```
var
uretici: array[0..3] of DWord;
x:PChar;
begin
form1.AlphaBlend:=true;
form1.AlphaBlendValue:=0;
timer1.Enabled:=false;
timer2.Enabled:=false;
timer3.Enabled:=false;
FORM1.Height:=0;
FORM1.Width:=760;
form1.statusbar1.Panels[3].Text:='User : NONE';
asm
  push ebx
  mov eax, 0
  dw $A20F
  mov DWord ptr uretici, ebx
  mov DWord ptr uretici[+4], edx
  mov DWord ptr uretici[+8], ecx
  pop ebx
end;
 uretici[3]:=0;
 x:=@uretici;
 statusbar1.Panels[4].Text:='CPU Type : '+x;
end;
```

4.1.3 Main menu buttons

This is we used which button in mainmenu form.

4.1.3.1 File

4.1.3.1.a New registration

```
begin
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
query2.Last;
w:=strtoint(query2.Fields[2].AsString);
e:=strtoint(query2.Fields[3].AsString);
r:=strtoint(query2.Fields[4].AsString);
d:=strtoint(query2.Fields[5].AsString);
if (w=1) then form2.dxButton2.Enabled:=true else form2.dxButton2.Enabled:=false;
if (e=1) then form2.dxButton3.Enabled:=true else form2.dxButton3.Enabled:=false;
if (d=1) then form2.dxButton4.Enabled:=true else form2.dxButton4.Enabled:=false;
if (r=1) then
begin
form2.dxButton8.Enabled:=true;
form2.DBGrid1.Enabled:=true;
end else begin
form2.dxButton8.Enabled:=false;
form2.DBGrid1.Enabled:=false;
end;
form2.show;
form1.enabled:=false;
end;
end;
```

4.1.3.1.b List

```
begin
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
query2.Last;
r:=strtoint(query2.Fields[4].AsString);
if (r=1) then
begin
 form3.dxButton4.Enabled:=true;
 form3.DBGrid1.Enabled:=true;
end else begin
 form3.dxButton4.Enabled:=false;
 form3.DBGrid1.Enabled:=false;
end;
```

4.1.3.1.c Search, edit and delete

```
Begin
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
query2.Last;
w:=strtoint(query2.Fields[2].AsString);
e:=strtoint(query2.Fields[3].AsString);
r:=strtoint(query2.Fields[4].AsString);
d:=strtoint(query2.Fields[5].AsString);
if (w=1) then form2.dxButton2.Enabled:=true else form2.dxButton2.Enabled:=false;
if (e=1) then form2.dxButton3.Enabled:=true else form2.dxButton3.Enabled:=false;
if (d=1) then form2.dxButton4.Enabled:=true else form2.dxButton4.Enabled:=false;
if (r=1) then
```

```
begin
form2.dxButton8.Enabled:=true;
form2.DBGrid1.Enabled:=true;
end else begin
form2.dxButton8.Enabled:=false;
form2.DBGrid1.Enabled:=false;
end;
form2.show;
form1.enabled:=false;
end;
end;
```

4.1.3.2 Inspection

4.1.3.2.a New register

Begin

```
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
form4.combobox1.items.clear;
form4.query2.first;
while not form4.query2.eof do begin
form4.combobox1.items.add(form4.query2.fields[0].asstring);
form4.query2.next;
end;
form4.query3.close;
query2.Last;
w:=strtoint(query2.Fields[2].AsString);
e:=strtoint(query2.Fields[3].AsString);
r:=strtoint(query2.Fields[4].AsString);
d:=strtoint(query2.Fields[5].AsString);
if (w=1) then form4.dxButton2.Enabled:=true else form4.dxButton2.Enabled:=false;
```

```
if (e=1) then form4.dxButton3.Enabled:=true else form4.dxButton3.Enabled:=false;
if (d=1) then form4.dxButton4.Enabled:=true else form4.dxButton4.Enabled:=false;
if (r=1) then form4.DBGrid1.Enabled:=true   else form4.DBGrid1.Enabled:=false;
form4.show;
form1.Enabled:=false;
end;
end;
```

4.1.3.2.b List

```
Begin
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
query2.Last;
r:=strtoint(query2.Fields[4].AsString);
if (r=1) then
begin
form5.ComboBox1.Enabled:=true;
form5.tfXPEdit4.Enabled:=true;
form5.tfXPEdit1.Enabled:=true;
form5.RadioGroup1.Enabled:=true;
form5.DBGrid1.Enabled:=true;
form5.DBGrid2.Enabled:=true;
end
else begin
form5.ComboBox1.Enabled:=false;
form5.tfXPEdit4.Enabled:=false;
form5.tfXPEdit1.Enabled:=false;
form5.RadioGroup1.Enabled:=false;
form5.DBGrid1.Enabled:=false;
form5.DBGrid2.Enabled:=false;
```

```
end;
form5.show;
form1.Enabled:=false;
end;
end;
```

4.1.3.2.c Delete, edit and search

```
begin
form4.combobox1.items.clear;
form4.query2.first;
while not form4.query2.eof do begin
form4.combobox1.items.add(form4.query2.fields[0].asstring);
form4.query2.next;
end;
if form1.statusbar1.panels[3].text='user : none' then begin
application.messagebox('there is no user login','warning!!!',16);
end
else begin
query2.last;
w:=strtoint(query2.fields[2].asstring);
e:=strtoint(query2.fields[3].asstring);
r:=strtoint(query2.fields[4].asstring);
d:=strtoint(query2.fields[5].asstring);
if (w=1) then form4.dxbutton2.enabled:=true else form4.dxbutton2.enabled:=false;
if (e=1) then form4.dxbutton3.enabled:=true else form4.dxbutton3.enabled:=false;
if (d=1) then form4.dxbutton4.enabled:=true else form4.dxbutton4.enabled:=false;
if (r=1) then form4.dbgrid1.enabled:=true   else form4.dbgrid1.enabled:=false;
form4.show;
form1.enabled:=false;
end;
end;
```

4.1.3.3 Appointment

4.1.3.3.a New appointment

Begin

```
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
query2.Last;
w:=strtoint(query2.Fields[2].AsString);
e:=strtoint(query2.Fields[3].AsString);
r:=strtoint(query2.Fields[4].AsString);
d:=strtoint(query2.Fields[5].AsString);
if (w=1) then form9.dxButton1.Enabled:=true else form9.dxButton1.Enabled:=false;
if (e=1) then form9.dxButton2.Enabled:=true else form9.dxButton2.Enabled:=false;
if (d=1) then form9.dxButton3.Enabled:=true else form9.dxButton3.Enabled:=false;
if (r=1) then
begin
form9.dxButton5.Enabled:=true;
form9.DBGrid1.Enabled:=true; end
else begin
form9.dxButton5.Enabled:=false;
form9.DBGrid1.Enabled:=false;
end;
form9.show;
form1.Enabled:=false;
end;
end;
```

4.1.3.3.b List

Begin

```
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
```

```
end
else begin
query2.Last;
w:=strtoint(query2.Fields[2].AsString);
e:=strtoint(query2.Fields[3].AsString);
r:=strtoint(query2.Fields[4].AsString);
d:=strtoint(query2.Fields[5].AsString);
if (r=1) then begin
form10.RadioGroup1.Enabled:=true;
form10.DBGrid1.Enabled:=true;
form10.dxButton1.Enabled:=true;
end else begin
form10.RadioGroup1.Enabled:=false;
form10.DBGrid1.Enabled:=false;
form10.dxButton1.Enabled:=false;
end;
form10.show;
form1.Enabled:=false;
end;
end;
```

4.1.3.3.c Delete, edit and search

```
Begin
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
query2.Last;
w:=strtoint(query2.Fields[2].AsString);
e:=strtoint(query2.Fields[3].AsString);
r:=strtoint(query2.Fields[4].AsString);
d:=strtoint(query2.Fields[5].AsString);
if (w=1) then form9.dxButton1.Enabled:=true else form9.dxButton1.Enabled:=false;
```

```
if (e=1) then form9.dxButton2.Enabled:=true else form9.dxButton2.Enabled:=false;
if (d=1) then form9.dxButton3.Enabled:=true else form9.dxButton3.Enabled:=false;
if (r=1) then
begin
form9.dxButton5.Enabled:=true;
form9.DBGrid1.Enabled:=true; end
else begin
form9.dxButton5.Enabled:=false;
form9.DBGrid1.Enabled:=false;
end;
form9.show;
form1.Enabled:=false;
end;
end;
```

4.1.3.4 New user register

```
Begin
query2.Last;
w:=strtoint(query2.Fields[2].AsString);
e:=strtoint(query2.Fields[3].AsString);
r:=strtoint(query2.Fields[4].AsString);
d:=strtoint(query2.Fields[5].AsString);
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
if (w=1)and(r=1)and(e=1)and(d=1) then begin
form14.show;
form1.Enabled:=false;
end else application.MessageBox('You are not authorized to use this part!!!','! ! ! W a r
n i n g ! ! !',16);
end;
```

4.1.3.5 Show appointment

```
Begin
if form1.statusbar1.Panels[3].Text='User : NONE' then begin
application.MessageBox('There is no user login','Warning!!!',16);
end
else begin
query2.Last;
r:=strtoint(query2.Fields[4].AsString);
if (r=1) then
begin
form1.Query1.SQL.Clear;
form1.Query1.SQL.Text:='select * from app where apdate like'+#39+(a1)+'%'+#39;
form1.Query1.Open;
timer5.Enabled:=true;
end else application.MessageBox('You are not authorized to use this part!!!','Warning!!!',16);
end;
end;
```

4.2 File register form

4.2.1 New register Codes

4.2.1.a Necessary functions

```
Function RS(PWLen: integer): string;
const StrTable: string =
  '123456789';
var
 N, K, X, Y: integer;
begin
 Randomize;
 if (PWlen > Length(StrTable)) then K := Length(StrTable)-1
```

```
  else K := PWLen;
 SetLength(result, K);
 Y := Length(StrTable);
 N := 0;
 while N < K do begin
  X := Random(Y) + 1;
  if (pos(StrTable[X], result) = 0) then begin
   inc(N);
   Result[N] := StrTable[X];
  end;
 end;
procedure clear;
begin
form2.edit1.Text:='';
form2.edit2.Text:='';
form2.edit3.Text:='';
form2.edit4.Text:='';
form2.edit5.Text:='';
form2.maskedit1.Text:='';
form2.maskedit2.Text:='';
form2.maskedit3.Text:='';
form2.AGE.Text:='';
form2.Memo1.Text:='';
end;
procedure save;
begin
form2.Query1.Fields[0].AsString:=form2.Edit1.Text;
form2.Query1.Fields[1].AsString:=form2.Edit2.Text;
form2.Query1.Fields[2].AsString:=form2.Edit3.Text;
form2.Query1.Fields[3].AsString:=form2.maskEdit1.Text;
form2.Query1.Fields[4].AsString:=form2.Edit4.Text;
form2.Query1.Fields[5].AsString:=form2.AGE.Text;
form2.Query1.Fields[6].AsString:=form2.Edit5.Text;
```

```
form2.Query1.Fields[7].AsString:=form2.tfXPEdit1.Text;
form2.Query1.Fields[8].AsString:=form2.memo1.Text;
form2.Query1.Fields[9].AsString:=form2.maskEdit2.Text;
form2.Query1.Fields[10].AsString:=form2.maskEdit3.Text;
end;
procedure read;
begin
form2.Edit1.Text:=form2.Query1.Fields[0].AsString;
form2.Edit2.Text:=form2.Query1.Fields[1].AsString;
form2.Edit3.Text:=form2.Query1.Fields[2].AsString;
form2.maskEdit1.Text:=form2.Query1.Fields[3].AsString;
form2.Edit4.Text:=form2.Query1.Fields[4].AsString;
form2.AGE.Text:=form2.Query1.Fields[5].AsString;
form2.Edit5.Text:=form2.Query1.Fields[6].AsString;
form2.tfXPEdit1.Text:=form2.Query1.Fields[7].AsString;
form2.memo1.Text:=form2.Query1.Fields[8].AsString;
form2.maskEdit2.Text:=form2.Query1.Fields[9].AsString;
form2.maskEdit3.Text:=form2.Query1.Fields[10].AsString;
end;
function src(a:string):boolean;
begin
src:=false;
form2.Query1.First;
while not form2.Query1.eof do
if (a=form2.Query1.Fields[0].asstring)then
begin
src:=true;
exit;
end
else
form2.Query1.Next;
end;
```

4.2.1.b Take FileID functions

```
var
b:string;
begin
edit1.Text:=FormatDateTime('yyyy',Date);
edit1.Text:=edit1.Text+RS(4);
b:=edit1.text;
src(b);
while (query1.Fields[0].AsString=b) do begin
edit1.Text:=FormatDateTime('yyyy',Date);
edit1.Text:=edit1.Text+RS(4);
end;
end;
```

4.2.1.c Save button

```
var
sr:word;
begin
src(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
application.MessageBox('The File is registered','Warning!!!',16);
read;
end
else begin
sr:=application.MessageBox('Are you sure?','Question?',36);
if (sr=İDYES) then begin
query1.İnsert;
save;
query1.Post;
clear;
```

```
query2.Refresh;
end;
```

4.2.1.d Edit button

```
var
sr:word;
begin
sr:=application.MessageBox('Are you sure?','Question?',36);
if (sr=İDYES) then begin
query1.edit;
save;
query1.Post;
clear;
query2.Refresh;
end;
end;
```

4.2.1.e Delete button

```
begin
src(edit1.Text);
if (query1.Fields[0].AsString=edit1.Text) then
begin
query1.Delete;
clear;
query2.Refresh;
end else application.MessageBox('There is no record','Warning!!!',16);
end;
```

4.2.1.f Search button

```
if (radiogroup1.itemindex=0) then begin
form2.Query2.SQL.Clear;
form2.Query2.SQL.Text:='select * from filereg where fileid like' + #39+
(form2.edit6.text)+'%'+#39;
form2.Query2.Open;
```

```
end;
if (radiogroup1.itemindex=1) then begin
form2.Query2.SQL.Clear;
form2.Query2.SQL.Text:='select * from filereg where name like'+#39+
(form2.edit6.text)+'%'+#39;
form2.Query2.Open;
end;
if (radiogroup1.itemindex=2) then begin
form2.Query2.SQL.Clear;
form2.Query2.SQL.Text:='select * from filereg where surname like'+#39+
(form2.edit6.text)+'%'+#39;
form2.Query2.Open;
end;
if (radiogroup1.itemindex=3) then begin
form2.Query2.SQL.Clear;
form2.Query2.SQL.Text:='select * from filereg where job like'+#39+
(form2.edit6.text)+'%'+#39;
form2.Query2.Open;
end;
if (radiogroup1.itemindex=4) then begin
form2.Query2.SQL.Clear;
form2.Query2.SQL.Text:='select * from filereg where sex like'+#39+
(form2.edit6.text)+'%'+#39;
form2.Query2.Open;
end;
```

4.2.1.g Dbgrid read double click

```
var
a:string;
begin
a:=dbgrid1.Fields[0].AsString;
src(a);
if(query1.Fields[0].AsString=a) then
begin
```

```
clear;
read;
end;
```

4.2.2 List

4.2.2.a Necessary functions

```
procedure read;
begin
form2.Edit1.Text:=form3.Query1.Fields[0].AsString;
form2.Edit2.Text:=form3.Query1.Fields[1].AsString;
form2.Edit3.Text:=form3.Query1.Fields[2].AsString;
form2.maskEdit1.Text:=form3.Query1.Fields[3].AsString;
form2.Edit4.Text:=form3.Query1.Fields[4].AsString;
form2.AGE.Text:=form3.Query1.Fields[5].AsString;
form2.Edit5.Text:=form3.Query1.Fields[6].AsString;
form2.tfXPEdit1.Text:=form3.Query1.Fields[7].AsString;
form2.memo1.Text:=form3.Query1.Fields[8].AsString;
form2.maskEdit2.Text:=form3.Query1.Fields[9].AsString;
form2.maskEdit3.Text:=form3.Query1.Fields[10].AsString;
end;
function src(a:string):boolean;
begin
src:=false;
form3.Query1.First;
while not form3.Query1.eof do
if (a=form3.Query1.Fields[0].asstring)then
begin
src:=true;
exit;
end
else
form3.Query1.Next;
end;
```

4.2.2.b Search button

```
if (radiogroup2.itemindex=0) then begin
form3.Query1.SQL.Clear;
form3.Query1.SQL.Text:='select * from filereg where fileid like'+#39+
(form3.tfXPEdit1.text)+'%'+#39;
form3.Query1.Open;
end;
if (radiogroup2.itemindex=1) then begin
form3.Query1.SQL.Clear;
form3.Query1.SQL.Text:='select * from filereg where name like'+#39+
(form3.tfXPEdit1.text)+'%'+#39;
form3.Query1.Open;
end;
if (radiogroup2.itemindex=2) then begin
form3.Query1.SQL.Clear;
form3.Query1.SQL.Text:='select * from filereg where surname
like'+#39+(form3.tfXPEdit1.text)+'%'+#39;
form3.Query1.Open;
end;
if (radiogroup2.itemindex=3) then begin
form3.Query1.SQL.Clear;
form3.Query1.SQL.Text:='select * from filereg where job like'+#39+
(form3.tfXPEdit1.text)+'%'+#39;
form3.Query1.Open;
end;
if (radiogroup2.itemindex=4) then begin
form3.Query1.SQL.Clear;
form3.Query1.SQL.Text:='select * from filereg where sex like'+#39+
(form3.tfXPEdit1.text)+'%'+#39;
form3.Query1.Open;
end;
```

4.2.2.c Print buttons

```
form16.QuickRep1.Print;
```

4.3 Patient

4.3.1 New register codes

4.3.1.a Necessary functions

```
procedure clear;
begin
form4.ComboBox1.Text:='';
form4.tfXPEdit1.Text:='';
form4.tfXPEdit2.Text:='';
form4.tfXPEdit3.Text:='';
form4.tfXPEdit4.Text:='';
form4.tfXPEdit5.Text:='';
form4.tfXPEdit6.Text:='';
form4.tfXPEdit8.Text:='';
form4.MaskEdit1.Text:='';
form4.MaskEdit2.Text:='';
form4.MaskEdit3.Text:='';
form4.Memo1.Text:='';
form4.Memo2.Text:='';
form4.Memo3.Text:='';
form4.Memo4.Text:='';
form4.Memo5.Text:='';
form4.Memo6.Text:='';
form4.Memo7.Text:='';
form4.Memo8.Text:='';
end;
procedure save;
begin
form4.Query1.Fields[0].AsString:=form4.ComboBox1.Text;
form4.Query1.Fields[1].AsString:=form4.tfXPEdit1.Text;
```

```
form4.Query1.Fields[2].AsString:=form4.tfXPEdit2.Text;
form4.Query1.Fields[3].AsString:=form4.tfXPEdit7.Text;
form4.Query1.Fields[4].AsString:=form4.tfXPEdit3.Text;
form4.Query1.Fields[5].AsString:=form4.tfXPEdit4.Text;
form4.Query1.Fields[6].AsString:=form4.tfXPEdit5.Text;
form4.Query1.Fields[7].AsString:=form4.tfXPEdit6.Text;
form4.Query1.Fields[8].AsString:=form4.MaskEdit1.Text;
form4.Query1.Fields[9].AsString:=form4.MaskEdit2.Text;
form4.Query1.Fields[10].AsString:=form4.MaskEdit3.Text;
form4.Query1.Fields[11].AsString:=form4.Memo1.Text;
form4.Query1.Fields[12].AsString:=form4.Memo2.Text;
form4.Query1.Fields[13].AsString:=form4.Memo3.Text;
form4.Query1.Fields[14].AsString:=form4.Memo4.Text;
form4.Query1.Fields[15].AsString:=form4.Memo5.Text;
form4.Query1.Fields[16].AsString:=form4.Memo6.Text;
form4.Query1.Fields[18].AsString:=form4.Memo7.Text;
form4.Query1.Fields[17].AsString:=form4.Memo8.Text;
end;
procedure read;
begin
form4.ComboBox1.Text:=form4.Query1.Fields[0].AsString;
form4.tfXPEdit1.Text:=form4.Query1.Fields[1].AsString;
form4.tfXPEdit2.Text:=form4.Query1.Fields[2].AsString;
form4.tfXPEdit7.Text:=form4.Query1.Fields[3].AsString;
form4.tfXPEdit3.Text:=form4.Query1.Fields[4].AsString;
form4.tfXPEdit4.Text:=form4.Query1.Fields[5].AsString;
form4.tfXPEdit5.Text:=form4.Query1.Fields[6].AsString;
form4.tfXPEdit6.Text:=form4.Query1.Fields[7].AsString;
form4.MaskEdit1.Text:=form4.Query1.Fields[8].AsString;
form4.MaskEdit2.Text:=form4.Query1.Fields[9].AsString;
form4.MaskEdit3.Text:=form4.Query1.Fields[10].AsString;
form4.Memo1.Text:=form4.Query1.Fields[11].AsString;
form4.Memo2.Text:=form4.Query1.Fields[12].AsString;
```

```
form4.Memo3.Text:=form4.Query1.Fields[13].AsString;
form4.Memo4.Text:=form4.Query1.Fields[14].AsString;
form4.Memo5.Text:=form4.Query1.Fields[15].AsString;
form4.Memo6.Text:=form4.Query1.Fields[16].AsString;
form4.Memo7.Text:=form4.Query1.Fields[18].AsString;
form4.Memo8.Text:=form4.Query1.Fields[17].AsString;
end;
function srcd(a:string;b:string):boolean;
begin
srcd:=false;
form4.Query1.First;
while not form4.Query1.eof do
if (a=form4.Query1.Fields[0].asstring) and (b=form4.Query1.Fields[3].asstring)then
begin
srcd:=true;
exit;
end
else
form4.Query1.Next;
end;
function src(a:string):boolean;
begin
src:=false;
form4.Query2.First;
while not form4.Query2.eof do
if (a=form4.Query2.Fields[0].asstring)then
begin
src:=true;
exit;
end
else
form4.Query2.Next;
end;
```

4.3.1.b Save button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=İDYES then begin
query1.İnsert;
save;
query1.Post;
query3.SQL.Clear;
query3.SQL.Text:='select * from sick where fileid like'+#39+
(combobox1.text)+'%'+#39;
query3.Open;
clear;
tfxpedit7.Text:=datetostr(date);
end;
```

4.3.1.c Edit button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=İDYES then begin
query1.edit;
save;
query1.Post;
query3.SQL.Clear;
query3.SQL.Text:='select * from sick where fileid
like'+#39+(combobox1.text)+'%'+#39;
query3.Open;
clear;
tfxpedit7.Text:=datetostr(date);
```

```
end;
```

4.3.1.d Delete button

```
var
a:word;
begin
src(combobox1.Text);
if (query1.Fields[0].AsString=combobox1.Text) then begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=İDYES then begin
query1.delete;
query3.SQL.Clear;
query3.SQL.Text:='select * from sick where fileid like'+#39+
(combobox1.text)+'%'+#39;
query3.Open;
clear;
tfxpedit7.Text:=datetostr(date);
end;
end;
end;
```

4.3.1.e FileID select

```
Begin
QUERY3.SQL.Clear;
query3.SQL.Text:='select * from sick where fileid like'+#39+
(combobox1.text)+'%'+#39;
query3.Open;
src(combobox1.Text);
if (query2.Fields[0].AsString=combobox1.Text) then begin
form4.tfXPEdit1.Text:=query2.Fields[1].AsString;
form4.tfXPEdit2.Text:=query2.Fields[2].AsString;
end;
```

4.3.1.f Dbgrid double click

```
var
e,d:string;
begin
e:=dbgrid1.fields[3].asstring;
d:=dbgrid1.fields[0].asstring;
srcd(d,e);
if (query1.fields[0].asstring=d) and (query1.fields[3].asstring=e) then begin
clear;
read;
end;
```

4.3.1.1 Laboratory register codes

4.3.1.1.a Necessary functions

```
procedure clear;
begin
form6.tfXPEdit4.Text:='';
form6.tfXPEdit5.Text:='';
form6.tfXPEdit6.Text:='';
form6.tfXPEdit7.Text:='';
form6.tfXPEdit8.Text:='';
form6.tfXPEdit9.Text:='';
form6.MaskEdit1.Text:='';
form6.MaskEdit2.Text:='';
form6.MaskEdit3.Text:='';
form6.MaskEdit4.Text:='';
form6.MaskEdit5.Text:='';
form6.MaskEdit6.Text:='';
form6.MaskEdit7.Text:='';
form6.Memo1.Text:='';
form6.Memo2.Text:='';
end;
```

```
procedure read;
begin
form6.tfXPEdit1.Text:=form6.Query1.Fields[0].AsString;
form6.tfXPEdit2.Text:=form6.Query1.Fields[1].AsString;
form6.tfXPEdit3.Text:=form6.Query1.Fields[2].AsString;
form6.tfXPEdit10.Text:=form6.Query1.Fields[3].AsString;
form6.MASKEdit5.Text:=form6.Query1.Fields[4].AsString;
form6.MASKEdit6.Text:=form6.Query1.Fields[5].AsString;
form6.tfXPEdit4.Text:=form6.Query1.Fields[6].AsString;
form6.tfXPEdit5.Text:=form6.Query1.Fields[7].AsString;
form6.tfXPEdit6.Text:=form6.Query1.Fields[8].AsString;
form6.tfXPEdit7.Text:=form6.Query1.Fields[9].AsString;
form6.tfXPEdit8.Text:=form6.Query1.Fields[10].AsString;
form6.maskEdit1.Text:=form6.Query1.Fields[11].AsString;
form6.maskEdit2.Text:=form6.Query1.Fields[12].AsString;
form6.maskEdit3.Text:=form6.Query1.Fields[13].AsString;
form6.maskEdit7.Text:=form6.Query1.Fields[14].AsString;
form6.maskEdit4.Text:=form6.Query1.Fields[15].AsString;
form6.tfXPEdit9.Text:=form6.Query1.Fields[16].AsString;
form6.memo1.Text:=form6.Query1.Fields[17].AsString;
form6.memo2.Text:=form6.Query1.Fields[18].AsString;
end;
procedure save;
begin
form6.Query1.Fields[0].AsString:=form6.tfXPEdit1.Text;
form6.Query1.Fields[1].AsString:=form6.tfXPEdit2.Text;
form6.Query1.Fields[2].AsString:=form6.tfXPEdit3.Text;
form6.Query1.Fields[3].AsString:=form6.tfXPEdit10.Text;
form6.Query1.Fields[4].AsString:=form6.MASKEdit5.Text;
form6.Query1.Fields[5].AsString:=form6.MASKEdit6.Text;
form6.Query1.Fields[6].AsString:=form6.tfXPEdit4.Text;
form6.Query1.Fields[7].AsString:=form6.tfXPEdit5.Text;
```

```
form6.Query1.Fields[8].AsString:=form6.tfXPEdit6.Text;
form6.Query1.Fields[9].AsString:=form6.tfXPEdit7.Text;
form6.Query1.Fields[10].AsString:=form6.tfXPEdit8.Text;
form6.Query1.Fields[11].AsString:=form6.maskEdit1.Text;
form6.Query1.Fields[12].AsString:=form6.maskEdit2.Text;
form6.Query1.Fields[13].AsString:=form6.maskEdit3.Text;
form6.Query1.Fields[14].AsString:=form6.maskEdit7.Text;
form6.Query1.Fields[15].AsString:=form6.maskEdit4.Text;
form6.Query1.Fields[16].AsString:=form6.tfXPEdit9.Text;
form6.Query1.Fields[17].AsString:=form6.memo1.Text;
form6.Query1.Fields[18].AsString:=form6.memo2.Text;
end;
function srcd(a:string;b:string):boolean;
begin
srcd:=false;
form6.Query1.First;
while not form6.Query1.eof do
if (a=form6.Query1.Fields[0].asstring) and (b=form6.Query1.Fields[3].asstring)then
begin
srcd:=true;
exit;
end
else
form6.Query1.Next;
end;
```

4.3.1.1.b Save button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=IDYES then begin
query1.Insert;
```

```
save;
query1.Post;
query1.SQL.Clear;
query1.SQL.Text:='select * from lab where fileid like'+#39+
(form6.tfXPEdit1.Text)+'%'+#39;
query1.Open;
clear;
end;
```

4.3.1.1.c Edit button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=IDYES then begin
query1.edit;
save;
query1.Post;
query1.SQL.Clear;
query1.SQL.Text:='select * from lab where fileid like'+#39+
(form6.tfXPEdit1.Text)+'%'+#39;
query1.open;
clear;
end;
```

4.3.1.1.d Delete button

```
var
k,l:string;
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=IDYES then begin
k:=form6.tfXPEdit1.Text;
l:=form6.tfXPEdit10.Text;
srcd(k,l);
```

```
if (form6.Query1.Fields[0].AsString=k) and (form6.Query1.Fields[3].AsString=l) then
begin
query1.Delete;
query1.SQL.Clear;
query1.SQL.Text:='select * from lab where fileid like'+#39+
(form6.tfXPEdit1.Text)+'%'+#39;
query1.open;
clear;
end;
end;
```

4.3.1.1.e Print button

```
Begin
form17.QRLabel24.Caption:=form6.tfXPEdit1.Text;
form17.QRLabel25.Caption:=form6.tfXPEdit2.Text;
form17.QRLabel26.Caption:=form6.tfXPEdit3.Text;
form17.QRLabel27.Caption:=form6.tfXPEdit10.Text;
form17.QRLabel28.Caption:=form6.maskedit5.Text;
form17.QRLabel29.Caption:=form6.maskEdit6.Text;
form17.QRLabel30.Caption:=form6.tfXPEdit4.Text;
form17.QRLabel31.Caption:=form6.tfXPEdit5.Text;
form17.QRLabel32.Caption:=form6.tfXPEdit6.Text;
form17.QRLabel33.Caption:=form6.tfXPEdit7.Text;
form17.QRLabel34.Caption:=form6.tfXPEdit8.Text;
form17.QRLabel35.Caption:=form6.maskEdit1.Text;
form17.QRLabel36.Caption:=form6.maskEdit2.Text;
form17.QRLabel37.Caption:=form6.maskEdit3.Text;
form17.QRLabel38.Caption:=form6.maskEdit7.Text;
form17.QRLabel39.Caption:=form6.maskEdit4.Text;
form17.QRLabel40.Caption:=form6.tfXPEdit9.Text;
form17.QRMemo1.Lines.Clear;
form17.QRMemo1.Lines.Text:=form6.Memo1.Text;
form17.QRMemo2.Lines.Clear;
```

```
form17.QRMemo2.Lines.Text:=form6.Memo2.Text;
form17.QuickRep1.Preview;
end;
```

4.3.1.1.f Dbgrid double click

```
var
k,l:string;
begin
k:=form6.DBGrid1.Fields[0].AsString;
l:=form6.DBGrid1.Fields[3].AsString;
srcd(k,l);
if (form6.Query1.Fields[0].AsString=k) and (form6.Query1.Fields[3].AsString=l) then
begin
read;
end;
```

4.3.1.2 Cure register codes

4.3.1.2.a Necesarry function

```
procedure clear;
begin
form7.memo1.Text:='';
form7.Memo2.Text:='';
end;
procedure save;
var
a:string;
begin
a:=datetostr(form7.DateTimePicker1.Date);
form7.Query1.Fields[0].AsString:=form7.tfXPEdit1.Text;
form7.Query1.Fields[1].AsString:=form7.tfXPEdit2.Text;
form7.Query1.Fields[2].AsString:=form7.tfXPEdit3.Text;
form7.Query1.Fields[3].AsString:=form7.tfXPEdit4.Text;
form7.Query1.Fields[4].AsString:=form7.tfXPEdit5.Text;
form7.Query1.Fields[5].AsString:=a;
```

```
form7.Query1.Fields[6].AsString:=form7.memo1.Text;
form7.Query1.Fields[7].AsString:=form7.memo2.Text;
end;
procedure read;
begin
form7.DateTimePicker1.DateTime:=strtodate(form7.Query1.Fields[5].AsString);
form7.tfXPEdit1.Text:=form7.Query1.Fields[0].AsString;
form7.tfXPEdit2.Text:=form7.Query1.Fields[1].AsString;
form7.tfXPEdit3.Text:=form7.Query1.Fields[2].AsString;
form7.tfXPEdit4.Text:=form7.Query1.Fields[3].AsString;
form7.tfXPEdit5.Text:=form7.Query1.Fields[4].AsString;
form7.memo1.Text:=form7.Query1.Fields[6].AsString;
form7.memo2.Text:=form7.Query1.Fields[7].AsString;
end;
function srcd(a:string;b:string):boolean;
begin
srcd:=false;
form7.Query1.First;
while not form7.Query1.eof do
if (a=form7.Query1.Fields[0].asstring) and (b=form7.Query1.Fields[3].asstring)then
begin
srcd:=true;
exit;
end
else
form7.Query1.Next;
end;
```

4.3.1.2.b Save button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
```

```
if a=IDYES then begin
query1.İnsert;
save;
query1.Post;
clear;
query1.Refresh;
form7.query1.SQL.Clear;
form7.query1.SQL.Text:='select * from cure where fileid like'+#39+
(form7.tfXPEdit1.Text)+'%'+#39;
form7.query1.open;
end;
```

4.3.1.2.c Edit button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=İDYES then begin
query1.Edit;
save;
query1.Post;
clear;
query1.Refresh;
form7.query1.SQL.Clear;
form7.query1.SQL.Text:='select * from cure where fileid
like'+#39+(form7.tfXPEdit1.Text)+'%'+#39;
form7.query1.open;
end;
end;
```

4.3.1.2.d Delete button

```
var
a:word;
begin
```

```
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=IDYES then begin
srcd(form7.tfXPEdit1.Text,form7.tfXPEdit4.Text);
if(query1.Fields[0].AsString=form7.tfXPEdit1.Text)and
(query1.Fields[3].AsString=form7.tfXPEdit4.Text)  then
begin
query1.Delete;
clear;
query1.Refresh;
form7.query1.SQL.Clear;
form7.query1.SQL.Text:='select  *  from  cure  where  fileid  like'+#39+
(form7.tfXPEdit1.Text)+'%'+#39;
form7.query1.open;
end;
```

4.3.1.2.d Search button

```
form7.query1.SQL.Clear;
form7.query1.SQL.Text:='select  *  from  cure  where  fileid  =' +#39+
(form7.tfXPEdit1.Text) +#39 'and medicaldiag like' +#39+ (form7.tfXPEdit6.Text)+
'%'+#39;
form7.query1.open;
```

4.3.1.3 Recipe register codes

4.3.1.3.a Necesarry function

```
procedure clear;
begin
form8.tfXPEdit6.Text:='';
form8.tfXPEdit7.Text:='';
form8.tfXPEdit8.Text:='';
form8.tfXPEdit9.Text:='';
form8.tfXPEdit10.Text:='';
form8.tfXPEdit11.Text:='';
form8.tfXPEdit12.Text:='';
form8.tfXPEdit13.Text:='';
```

```
form8.tfXPEdit14.Text:='';
form8.tfXPEdit15.Text:='';
form8.tfXPEdit16.Text:='';
form8.tfXPEdit16.Text:='';
form8.tfXPEdit17.Text:='';
form8.tfXPEdit18.Text:='';
form8.tfXPEdit19.Text:='';
end;
procedure save;
begin
form8.Query1.Fields[0].AsString:=form8.tfXPEdit1.Text;
form8.Query1.Fields[1].AsString:=form8.tfXPEdit2.Text;
form8.Query1.Fields[2].AsString:=form8.tfXPEdit3.Text;
form8.Query1.Fields[3].AsString:=form8.tfXPEdit6.Text;
form8.Query1.Fields[4].AsString:=form8.tfXPEdit7.Text;
form8.Query1.Fields[5].AsString:=form8.tfXPEdit8.Text;
form8.Query1.Fields[6].AsString:=form8.tfXPEdit9.Text;
form8.Query1.Fields[7].AsString:=form8.tfXPEdit10.Text;
form8.Query1.Fields[8].AsString:=form8.tfXPEdit11.Text;
form8.Query1.Fields[9].AsString:=form8.tfXPEdit12.Text;
form8.Query1.Fields[10].AsString:=form8.tfXPEdit13.Text;
form8.Query1.Fields[11].AsString:=form8.tfXPEdit14.Text;
form8.Query1.Fields[12].AsString:=form8.tfXPEdit15.Text;
form8.Query1.Fields[13].AsString:=form8.tfXPEdit16.Text;
form8.Query1.Fields[14].AsString:=form8.tfXPEdit17.Text;
form8.Query1.Fields[15].AsString:=form8.tfXPEdit18.Text;
form8.Query1.Fields[16].AsString:=form8.tfXPEdit19.Text;
form8.Query1.Fields[17].AsString:=form8.tfXPEdit4.Text;
form8.Query1.Fields[18].AsString:=form8.tfXPEdit5.Text;
end;
Procedure read;
begin
form8.tfXPEdit1.Text:=form8.Query1.Fields[0].AsString;
```

```
form8.tfXPEdit2.Text:=form8.Query1.Fields[1].AsString;
form8.tfXPEdit3.Text:=form8.Query1.Fields[2].AsString;
form8.tfXPEdit6.Text:=form8.Query1.Fields[3].AsString;
form8.tfXPEdit7.Text:=form8.Query1.Fields[4].AsString;
form8.tfXPEdit8.Text:=form8.Query1.Fields[5].AsString;
form8.tfXPEdit9.Text:=form8.Query1.Fields[6].AsString;
form8.tfXPEdit10.Text:=form8.Query1.Fields[7].AsString;
form8.tfXPEdit11.Text:=form8.Query1.Fields[8].AsString;
form8.tfXPEdit12.Text:=form8.Query1.Fields[9].AsString;
form8.tfXPEdit13.Text:=form8.Query1.Fields[10].AsString;
form8.tfXPEdit14.Text:=form8.Query1.Fields[11].AsString;
form8.tfXPEdit15.Text:=form8.Query1.Fields[12].AsString;
form8.tfXPEdit16.Text:=form8.Query1.Fields[13].AsString;
form8.tfXPEdit17.Text:=form8.Query1.Fields[14].AsString;
form8.tfXPEdit18.Text:=form8.Query1.Fields[15].AsString;
form8.tfXPEdit19.Text:=form8.Query1.Fields[16].AsString;
form8.tfXPEdit4.Text:=form8.Query1.Fields[17].AsString;
form8.tfXPEdit5.Text:=form8.Query1.Fields[18].AsString;
end;
function srcd(a:string;b:string):boolean;
begin
srcd:=false;
form8.Query1.First;
while not form8.Query1.eof do
if (a=form8.Query1.Fields[0].asstring) and (b=form8.Query1.Fields[17].asstring)then
begin
srcd:=true;
exit;
end
else
form8.Query1.Next;
end;
```

4.3.1.3.b Save button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=IDYES then begin
query1.İnsert;
save;
query1.Post;
query1.Refresh;
form8.query1.SQL.Clear;
form8.query1.SQL.Text:='select * from recipe where fileid like' +#39+
(form8.tfXPEdit1.Text) +'%'+#39;
form8.query1.open;
clear;
end;
```

4.3.1.3.c Edit button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=IDYES then begin
query1.Edit;
save;
query1.Post;
query1.Refresh;
form8.query1.SQL.Clear;
form8.query1.SQL.Text:='select * from recipe where fileid like'+#39+
(form8.tfXPEdit1.Text)+'%'+#39;
form8.query1.open;
clear;
end;
```

4.3.1.3.d Delete button

```
var
c,d:string;
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=İDYES then begin
c:=form8.tfXPEdit1.Text;
d:=form8.tfXPEdit4.Text;
srcd(c,d);
if (form8.Query1.Fields[0].AsString=c)and (form8.Query1.Fields[17].AsString=d) then
begin
FORM8.Query1.Delete;
clear;
form8.Query1.Refresh;
form8.query1.SQL.Clear;
form8.query1.SQL.Text:='select * from recipe where fileid like'+#39+
(form8.tfXPEdit1.Text)+'%'+#39;
form8.query1.open;
end;
end;
```

4.3.1.3.e Dbgrid double click

```
var
c,d:string;
begin
c:=form8.tfXPEdit1.Text;
d:=form8.tfXPEdit4.Text;
srcd(c,d);
if (form8.Query1.Fields[0].AsString=c)and (form8.Query1.Fields[17].AsString=d) then
begin
read;
end;
```

4.3.1.3.f Print buttons

```
var
a,b,c,d:string;
begin
query2.Last;
a:=query2.Fields[3].AsString;
b:=query2.Fields[1].AsString;
c:=query2.Fields[2].AsString;
d:=a+' '+b+' '+c;
form18.QRLabel1.Caption:=d;
FORM18.QRLabel2.Caption:=query2.Fields[5].AsString;
FORM18.QRLabel3.Caption:=query2.Fields[4].AsString;
form18.QRLabel12.Caption:=form8.tfXPEdit1.Text;
form18.QRLabel13.Caption:=form8.tfXPEdit2.Text;
form18.QRLabel14.Caption:=form8.tfXPEdit3.Text;
form18.QRLabel15.Caption:=form8.tfXPEdit4.Text;
form18.QRLabel16.Caption:=form8.tfXPEdit5.Text;
form18.QRLabel20.Caption:=form8.tfXPEdit6.Text+' '+form8.tfXPEdit7.Text;
form18.QRLabel21.Caption:=form8.tfXPEdit8.Text+' '+form8.tfXPEdit9.Text;
form18.QRLabel22.Caption:=form8.tfXPEdit10.Text+' '+form8.tfXPEdit11.Text;
form18.QRLabel23.Caption:=form8.tfXPEdit12.Text+' '+form8.tfXPEdit13.Text;
form18.QRLabel24.Caption:=form8.tfXPEdit14.Text+' '+form8.tfXPEdit15.Text;
form18.QRLabel25.Caption:=form8.tfXPEdit16.Text+' '+form8.tfXPEdit17.Text;
form18.QRLabel26.Caption:=form8.tfXPEdit18.Text+' '+form8.tfXPEdit19.Text;
FORM18.qrlabel33.caption:=query2.Fields[8].AsString;
FORM18.qrlabel34.caption:=query2.Fields[9].AsString;
FORM18.qrlabel35.caption:=query2.Fields[7].AsString;
FORM18.qrmemo1.lines.text:=query2.Fields[6].AsString;
form18.QuickRep1.Preview;
end;
```

4.4 Appointment

4.4.1 New registration

4.4.1.a Necesarry function

```
procedure read;
begin
form9.tfXPEdit1.Text:=form9.Query1.Fields[0].AsString;
form9.tfXPEdit2.Text:=form9.Query1.Fields[1].AsString;
form9.tfXPEdit3.Text:=form9.Query1.Fields[2].AsString;
form9.tfXPEdit5.Text:=form9.Query1.Fields[3].AsString;
form9.DateTimePicker1.Date:=strtodate(form9.Query1.Fields[4].AsString);
form9.maskEdit1.Text:=form9.Query1.Fields[5].AsString;
end;
procedure save;
begin
form9.Query1.Fields[0].AsString:=form9.tfXPEdit1.Text;
form9.Query1.Fields[1].AsString:=form9.tfXPEdit2.Text;
form9.Query1.Fields[2].AsString:=form9.tfXPEdit3.Text;
form9.Query1.Fields[3].AsString:=form9.tfXPEdit5.Text;
form9.Query1.Fields[4].AsString:=datetostr(form9.DateTimePicker1.Date);
form9.Query1.Fields[5].AsString:=form9.maskEdit1.Text;
end;
procedure clear;
begin
form9.tfXPEdit1.Text:='';
form9.tfXPEdit2.Text:='';
form9.tfXPEdit3.Text:='';
form9.maskedit1.Text:='';
end;
function srcd(a:string;b:string):boolean;
begin
srcd:=false;
form9.Query1.First;
```

```
while not form9.Query1.eof do
if (a=form9.Query1.Fields[0].asstring) and (b=form9.Query1.Fields[1].asstring)then
begin
srcd:=true;
exit;
end
else
form9.Query1.Next;
end;
```

4.4.1.b Save button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=IDYES then begin
FORM9.tfXPEdit5.Text:=datetostr(date);
query1.İnsert;
save;
query1.Post;
clear;
query1.Refresh;
end;
```

4.4.1.c Edit button

```
var
a:word;
begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=İDYES then begin
FORM9.tfXPEdit5.Text:=datetostr(date);
query1.edit;
save;
query1.Post;
```

```
clear;
query1.Refresh;
end;
end;
```

4.4.1.d Delete button

```
var
a:word;
begin
srcd(form9.tfXPEdit1.Text,form9.tfXPEdit2.Text);
if(query1.Fields[0].AsString=form9.tfXPEdit1.Text)and
(query1.Fields[1].AsString=form9.tfXPEdit2.Text)then begin
a:=application.MessageBox('A r e  y o u  s u r e ?','!!!! W a r n i n g !!!!',36);
if a=İDYES then begin
query1.Delete;
clear;
FORM9.tfXPEdit5.Text:=datetostr(date);
query1.Refresh;
end;
end else application.MessageBox('There is no register....','!!!! W a r n i n g !!!!', 16);
end;
end;
```

4.4.1.e Search and read button

```
if radiogroup1.itemindex=0 then begin
form9.Query1.SQL.Clear;
form9.Query1.SQL.Text:='select * from app where name
like'+#39+(form9.tfXPEdit6.text)+'%'+#39;
form9.Query1.Open;
end;
if radiogroup1.itemindex=1 then begin
form9.Query1.SQL.Clear;
form9.Query1.SQL.Text:='select * from app where surname
like'+#39+(form9.tfXPEdit6.text)+'%'+#39;
```

```
form9.Query1.Open;
end;
```

4.4.2 List

4.4.2.a Necesarry function

```
procedure read;
begin
form9.tfXPEdit1.Text:=form10.Query1.Fields[0].AsString;
form9.tfXPEdit2.Text:=form10.Query1.Fields[1].AsString;
form9.tfXPEdit3.Text:=form10.Query1.Fields[2].AsString;
form9.tfXPEdit5.Text:=form10.Query1.Fields[3].AsString;
form9.DateTimePicker1.Date:=strtodate(form10.Query1.Fields[4].AsString);
form9.maskEdit1.Text:=form10.Query1.Fields[5].AsString;
end;
function srcd(a:string;b:string):boolean;
begin
srcd:=false;
form10.Query1.First;
while not form10.Query1.eof do
if (a=form10.Query1.Fields[0].asstring) and (b=form10.Query1.Fields[1].asstring)then
begin
srcd:=true;
exit;
end
else
form10.Query1.Next;
end;
```

4.4.2.b Seacrh button

```
form10.Query1.SQL.Clear;
form10.Query1.SQL.Text:='select * from app where apdate like'+#39+
(form10.tfXPEdit1.text)+'%'+#39;
```

```
form10.Query1.Open;
```

4.4.2.c List catagory

```
if radiogroup1.ItemIndex=0 then begin
query1.SQL.Clear;
query1.SQL.Text:='select * from app order by name ASC';
query1.Open;
end;
if radiogroup1.ItemIndex=1 then begin
query1.SQL.Clear;
query1.SQL.Text:='select * from app order by surname ASC';
query1.Open;
end;
if radiogroup1.ItemIndex=2 then begin
query1.SQL.Clear;
query1.SQL.Text:='select * from app order by regdate ASC';
query1.Open;
end;
if radiogroup1.ItemIndex=3 then begin
query1.SQL.Clear;
query1.SQL.Text:='select * from app order by apdate ASC';
query1.Open;
end;
if radiogroup1.ItemIndex=4 then begin
query1.SQL.Clear;
query1.SQL.Text:='select * from app order by aptime ASC';
query1.Open;
end;
```

4.4.2.d Dbgrid double click

```
var
w,e,d,r:integer;
begin
srcd(form10.DBGrid1.Fields[0].AsString,form10.DBGrid1.Fields[1].AsString);
```

```
if(query1.Fields[0].AsString=form10.DBGrid1.Fields[0].AsString)and(query1.Fields[1]
.AsString=form10.DBGrid1.Fields[1].AsString)then begin
read;
form1.query2.Last;
w:=strtoint(form1.query2.Fields[2].AsString);
e:=strtoint(form1.query2.Fields[3].AsString);
r:=strtoint(form1.query2.Fields[4].AsString);
d:=strtoint(form1.query2.Fields[5].AsString);
if (w=1) then form9.dxButton1.Enabled:=true else form9.dxButton1.Enabled:=false;
if (e=1) then form9.dxButton2.Enabled:=true else form9.dxButton2.Enabled:=false;
if (d=1) then form9.dxButton3.Enabled:=true else form9.dxButton3.Enabled:=false;
if (r=1) then
begin
form9.dxButton5.Enabled:=true;
form9.DBGrid1.Enabled:=true; end
else begin
form9.dxButton5.Enabled:=false;
form9.DBGrid1.Enabled:=false;
end;
```

4.5 License register

4.5.a Necessary function

```
*** Take  ProductID  using by HDD and CPU serial number ****
function GetideSerialNumber ():string;
Const IDENTIFY_BUFFER_SIZE = 512;
type
TIDERegs = packed record
bFeaturesReg     : BYTE;
bSectorCountReg  : BYTE;
 bSectorNumberReg : BYTE;
 bCylLowReg       : BYTE;
 bCylHighReg      : BYTE;
```

```
bDriveHeadReg    : BYTE;
bCommandReg      : BYTE;
bReserved        : BYTE;
end;
TSendCmdİnParams = packed record
cBufferSize  : DWORD;
irDriveRegs  : TİDERegs;
bDriveNumber : BYTE;
bReserved    : Array[0..2] of Byte;
dwReserved   : Array[0..3] of DWORD;
bBuffer      : Array[0..0] of Byte;
end;
TidSector = packed record
wGenConfig          : Word;
wNumCyls            : Word;
wReserved           : Word;
wNumHeads           : Word;
wBytesPerTrack      : Word;
wBytesPerSector     : Word;
wSectorsPerTrack    : Word;
wVendorUnique       : Array[0..2] of Word;
sSerialNumber       : Array[0..19] of CHAR;
wBufferType         : Word;
wBufferSize         : Word;
wECCSize            : Word;
sFirmwareRev        : Array[0..7] of Char;
sModelNumber        : Array[0..39] of Char;
wMoreVendorUnique   : Word;
wDoubleWordİO       : Word;
wCapabilities       : Word;
wReserved1          : Word;
wPIOTiming          : Word;
wDMATiming          : Word;
```

```
wBS                    : Word;
wNumCurrentCyls        : Word;
wNumCurrentHeads       : Word;
wNumCurrentSectorsPerTrack : Word;
ulCurrentSectorCapacity    : DWORD;
wMultSectorStuff       : Word;
ulTotalAddressableSectors  : DWORD;
wSingleWordDMA         : Word;
wMultiWordDMA          : Word;
bReserved              : Array[0..127] of BYTE;
end;
PidSector = ^TİdSector;
TDriverStatus = packed record
bDriverError : Byte;
bIDEStatus   : Byte;
bReserved    : Array[0..1] of Byte;
dwReserved   : Array[0..1] of DWORD;
end;
TSendCmdOutParams = packed record
cBufferSize  : DWORD;
DriverStatus : TDriverStatus;
bBuffer      : Array[0..0] of BYTE;
end;
var
hDevice : THandle;
cbBytesReturned : DWORD;
ptr : PChar;
SCIP : TSendCmdİnParams;
aIdOutCmd : Array [0..(SizeOf(TSendCmdOutParams)+iDENTiFY_BUFFER_SiZE-
1)-1] of Byte;
İdOutCmd  : TSendCmdOutParams absolute aİdOutCmd;
procedure ChangeByteOrder( var Data; Size : İnteger );
        var
```

```
ptr : PChar;
i : İnteger;
        c : Char;
begin
        ptr := @Data;
        for i := 0 to (Size shr 1)-1 do
        begin
        c := ptr^;
        ptr^ := (ptr+1)^;
        (ptr+1)^ := c;
         İnc(ptr,2);
        end;
        end;
begin
        Result := '';
        if SysUtils.Win32Platform=VER_PLATFORM_WİN32_NT then
 begin
hDevice := CreateFile( '\\.\PhysicalDrive0', GENERİC_READ or GENERİC_WRİTE,
        FİLE_SHARE_READ or FİLE_SHARE_WRİTE, nil, OPEN_EXİSTİNG, 0, 0
);
        end
        else
        hDevice := CreateFile( '\\.\SMARTVSD', 0, 0, nil, CREATE_NEW, 0, 0 );
        if hDevice=İNVALİD_HANDLE_VALUE then Exit;
        try
        FillChar(SCİP,SizeOf(TSendCmdİnParams)-1,#0);
        FillChar(aİdOutCmd,SizeOf(aİdOutCmd),#0);
        cbBytesReturned := 0;
        with SCİP do
        begin
        cBufferSize  := İDENTİFY_BUFFER_SİZE;
        with irDriveRegs do
        begin
```

```
        bSectorCountReg  := 1;
        bSectorNumberReg := 1;
        bDriveHeadReg    := $A0;
        bCommandReg      := $EC;
        end;
        end;
        if not DeviceİoControl( hDevice, $0007c088, @SCİP, S
        sizeOf(TSendCmdİnParams)-1,
@aİdOutCmd, SizeOf(aİdOutCmd), cbBytesReturned, nil ) then Exit;
        Finally
        CloseHandle(hDevice);
        end;
with PİdSector(@İdOutCmd.bBuffer)^ do
        begin
        ChangeByteOrder( sSerialNumber, SizeOf(sSerialNumber) );
        (PChar(@sSerialNumber)+SizeOf(sSerialNumber))^ := #0;
        Result := PChar(@sSerialNumber);
        end;
end;
procedure clear;
begin
form11.tfXPEdit2.Text:='';
form11.tfXPEdit3.Text:='';
form11.tfXPEdit5.Text:='';
form11.tfXPEdit6.Text:='';
form11.maskedit1.Text:='';
form11.maskedit2.Text:='';
form11.maskedit3.Text:='';
form11.memo1.Text:='';
end;
```

4.5.b Save button function

```
begin
form11.Query2.Last;
if (form11.Query2.Fields[0].AsString='1')then begin
form12.show;
form11.enabled:=false;
end;
end;
```

4.6 Serial register

4.6.a Necessary function

```
        //RS1
function RS1(PWLen: integer): string;
const StrTable: string =
   'ABCDE';
var
  N, K, X, Y: integer;
begin
  Randomize;
  if (PWlen > Length(StrTable)) then K := Length(StrTable)-1
    else K := PWLen;
  SetLength(result, K);
  Y := Length(StrTable);
  N := 0;
 while N < K do begin
  X := Random(Y) + 1;
    if (pos(StrTable[X], result) = 0) then begin
      inc(N);
      Result[N] := StrTable[X];
    end;
  end;
end;
```

```
//RS2
function RS2(PWLen: integer): string;
const StrTable: string =
   '12345';
var
  N, K, X, Y: integer;
begin
  Randomize;
  if (PWlen > Length(StrTable)) then K := Length(StrTable)-1
    else K := PWLen;
  SetLength(result, K);
  Y := Length(StrTable);
  N := 0;
  while N < K do begin
    X := Random(Y) + 1;
    if (pos(StrTable[X], result) = 0) then begin
      inc(N);
      Result[N] := StrTable[X];
    end;
  end;
end;
//RS3
function RS3(PWLen: integer): string;
const StrTable: string =
   'FGHJK';
var
  N, K, X, Y: integer;
begin
  Randomize;
  if (PWlen > Length(StrTable)) then K := Length(StrTable)-1
    else K := PWLen;
  SetLength(result, K);
  Y := Length(StrTable);
```

```
  N := 0;
  while N < K do begin
    X := Random(Y) + 1;
    if (pos(StrTable[X], result) = 0) then begin
      inc(N);
      Result[N] := StrTable[X];
    end;
  end; end;
 ///RS4
function RS4(PWLen: integer): string;
const StrTable: string =
    '67890';
var
  N, K, X, Y: integer;
begin
  Randomize;
  if (PWlen > Length(StrTable)) then K := Length(StrTable)-1
    else K := PWLen;
  SetLength(result, K);
  Y := Length(StrTable);
  N := 0;
  while N < K do begin
    X := Random(Y) + 1;
    if (pos(StrTable[X], result) = 0) then begin
      inc(N);
      Result[N] := StrTable[X];
    end;
  end;
end;
```

4.6.b Take new key

```
form12.tfXPEdit2.Text:=RS1(4);
form12.tfXPEdit3.Text:=RS2(4);
```

```
form12.tfXPEdit4.Text:=RS3(4);
form12.tfXPEdit5.Text:=RS4(4);
```

4.6.c Register

```
var
a,b:integer;
f,g:integer;
s1,s2:string;
begin
  a:=strtoint(form12.tfXPEdit3.text);
  b:=strtoint(form12.tfXPEdit5.text);
  f:=abs(ceil(a div 3));
  g:=abs(ceil(b div 3));
   s1:=inttostr(f);
   s2:=inttostr(g);
  if (form12.tfXPEdit8.text=s1) and (form12.tfXPEdit6.text=s2) then begin
  form12.Query1.edit;
  saveserial;
  form12.Query1.Post;
  form11.Query1.edit;
  savelicense;
  form11.Query1.Post;
  showmessage('key correct');
  form13.Close;
  form1.Close;
  form11.close;
  form12.close;
  end else
  showmessage('key not correct');
  form13.Close;
  form1.Close;
  form11.close;
  form12.close;   end;
```

CONCLUSION

After making so many researches about Delphi Programming language and investigating through internet to make this project, I learned many things about Delphi, because I obliged to finish my project and everything had to be done by myself alone. So making practical things is much better than learning it literary.

During the project I faced so many problems they were difficult for me because it was my first program, later after practicing and learning from books it became easier by the time and I used to know how to use Delphi and how to write codes. So the first 2 or 3 forms where hard to organize and write codes, but later other forms become easier in design and writing codes.

In the future other options could be add to the program, it can be updated according to the need, also it can be connected to the internet, at that time sales could be done on net, for instance when someone wants to buy something, he/she looks to the internet first, investigates about that item, its image, size, color and price, also the payment facilities could be shown to the customer, so if the customer likes what he/she wants to buy, he orders through internet and the workplace provides that item for him in the limited time.

REFERENCES

[1]. www.google.com.tr

[2]. www.delphiturk.com.tr

[3]. www.programlama.com

[4]. İhsan Karagülle and Zeytin Pala Borland Delphi 6.0

[5]. Marco Cantu Borland Delphi 5.0 with Object Oriented Pascal (Fifth Edition)

[6]. Faruk Çubukçu SQL Server 7.0 (First Edition)