NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Computer Part Sales Company

Graduation Project

COM- 400

Student: Hüseyin AR (20010552)

Supervisor: Asst. Prof. Dr. Firudin Muradov

Nicosia – 2006

ACKNOWLEDGEMENTS

First of all, I want to pay my regards and to express my sincere gratitude to my supervisor Assist. Prof. Dr. Firudin Muradov and all people who have contributed in the preparation of my project to complete it successfully. I would like to thank to my family who gave their lasting encouragement in my studies and enduring these all expenses and supporting me in all events, so that I could be successful in my life time. I specially thank to my mother whose prays have helped me to keep safe from every dark region of life.

I am also very much grateful to all my friends and colleagues who gave their precious time to help me and giving me their ever devotion and all valuable information which I really need to complete my project.

Further I am thankful to Near East University academic staff and all those people who helped me or encouraged me in completion of my project.

TABLE OF CONTENTS

ABSTRACT

Data, gathered around us as a collection of facts, is of no use unless it is organized and represented in some meaningful form. Data represented in some meaningful form like, tables, charts, or graphs become information, which can be easily processed. The collection of data, usually refereed to as the database, contains information about one particular enterprise. These days' databases are used by a variety of users and organizations, which are important tools in processing DBMS, which are designed to manage large amount of data.

This project has as its goal to develop software, processing information about activities of a computer-part sales company. Software developed in this project contains both employee information, and information associated with sales and purchase of computer parts. The project can be developed by improving the software for processing all activities of the company.

INTRODUCTION

Nowadays the technology is developed a lot and started to use by anyone in the world no matter who he/she is. Because of the technology is entered to every platform of our life human needed to combine both software and hardware. Without software the machines are nothing. They need software to operate.

The automation is also became a part of our lives. The people operate with automation systems in everywhere. This project is Computer Shop Stock Program. This Automation is used to keep the information about the product. The products are being kept in the stock according to the personnel.

In this project the main point is making the user's job easy. It lets to the manager to manage the personnel, customer information easily. And we he/she can get the sale report specific date.

CHAPTER 1

1. BASIC CONCEPT OF DELPHI 7

1.1. Introduction

Let us start an overview of the Delphi development environment to get you started using the product right away. It also tells you where to look for details about the tools and features available in Delphi.

You will be shown a tour of the environment describes the main tools on the Delphi desktop, or integrated desktop environment (IDE). It will be explained Programming with Delphi, explains how you use some of these tools to create an application. And also Customizing the desktop describes how you can customize the Delphi IDE for your development needs.

1.2. What is Delphi?

Delphi is an object-oriented, visual programming environment for rapid application development (RAD). Using Delphi, you can create highly efficient applications for Microsoft Windows XP, Microsoft Windows 2000 and Microsoft Windows 98 with a minimum of manual coding. Delphi also provides a simple cross-platform solution when used in conjunction with Kylix, Borland's RAD tool for Linux. 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.

1.2.1. **Registering Delphi**

Delphi can be registered in several ways. The first time you launch Delphi after installation, you will be prompted to enter your serial number and authorization key. Once this has been entered, a registration dialog offers four choices:

• Register using your internet connection.

Use this option to register online using your existing internet connection.

• Register by phone or Web browser.

Use this option to register by phone or through your web browser. If you received an activation key via email, use this option to select the file.

• Import software activation information from a file or email.

• Register later.

Online registration is the easiest way to register Delphi, but it requires that you have an active connection to the internet. If you are already a member of the Borland Community, or have an existing software registration account, simply enter the relevant account information. This will automatically register Delphi. If not, the registration process provides a way to create an account.

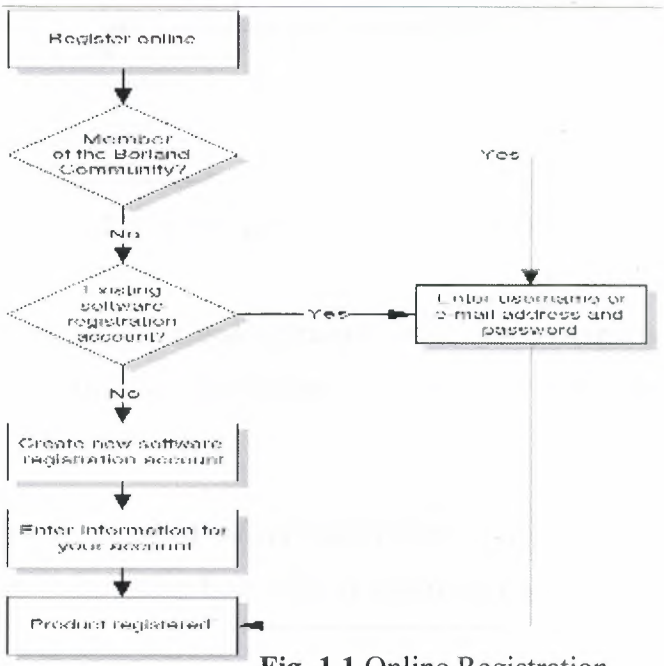


Fig. 1.1 Online Registration

The second option (register by phone or Web page) is useful if the machine you are installing on is not connected to the internet, or if you are behind a firewall that is blocking online registration.

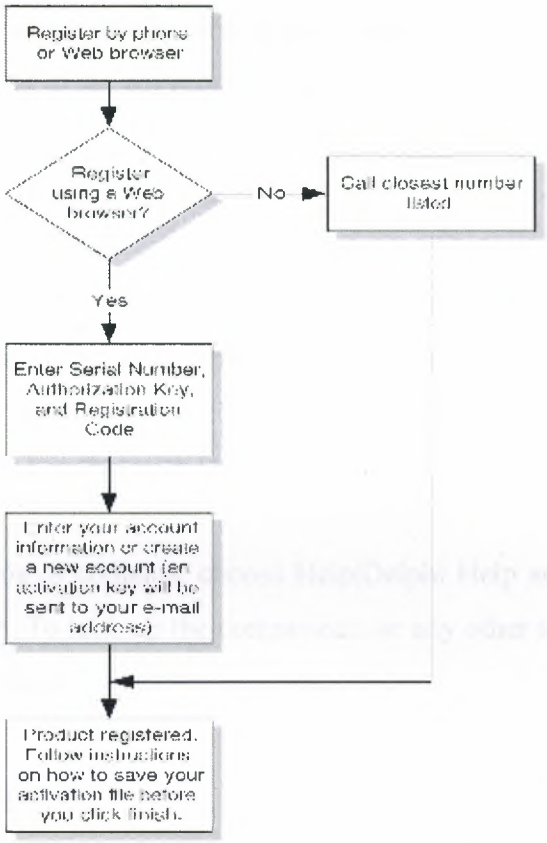


Fig. 1.2 Register by Phone or Web Page

If you have previously received software activation information, you can select theImport software activation information from a file or email option and select the activation.slip file on your system.

If you have previously received software activation information, you can select the Import software activation information from a file or email option and select the activation.slip file on your system.

1.2.2. Finding Information

You can find information on Delphi in the following ways:

• Online Help

• Printed documentation

• Borland developer support services and Web site

For information about new features in this release, refer to What's New in the online Help Contents and to the www.borland.com Web site.

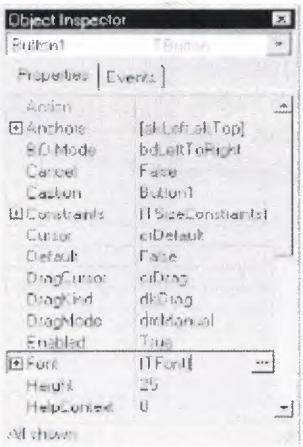
1.2.3. Online Help

The online Help system provides detailed information about user interface features, language implementation, programming tasks, and the components. It includes all the material in the Delphi Developer's Guide, Delphi Language Guide, and a host of Help files for other features bundled with Delphi.

To view the table of contents, choose Help|Delphi Help and Help|Delphi Tools, and click the Contents tab. To look up the components or any other topic, click the Index or Find tab and type your request.

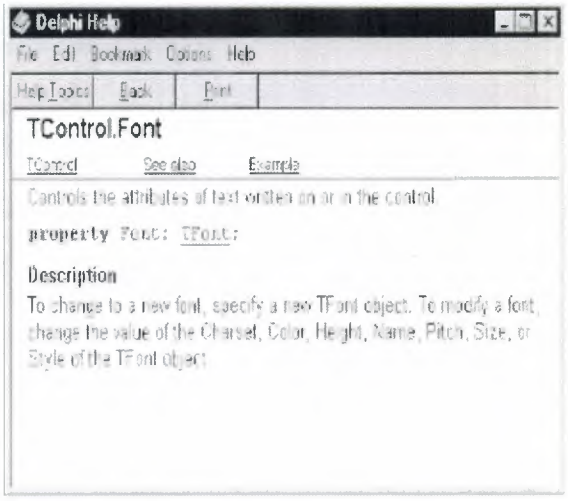
1.2.4. F1 Help

You can get context-sensitive Help on any part of the development environment, including menu items, dialog boxes, toolbars, and components by selecting the item and pressing F1.



Press F1 on a property or event name in the Object Inspector to display the VCL Help.

Fig. 1.3 TControl Font



In the Code editor, press F1 on a language, VCL, or CLX element.

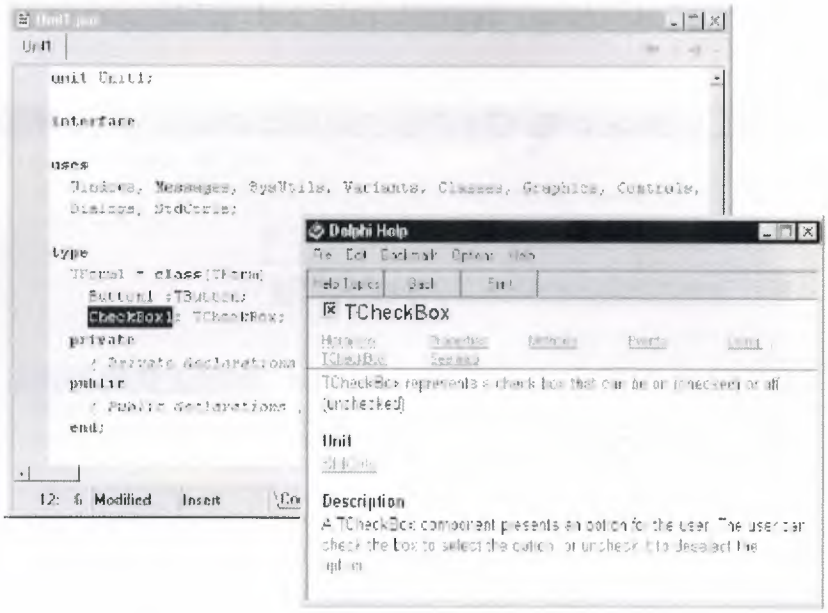


Fig. 1.4 TCheckBox

Press F1 on a component on a form.

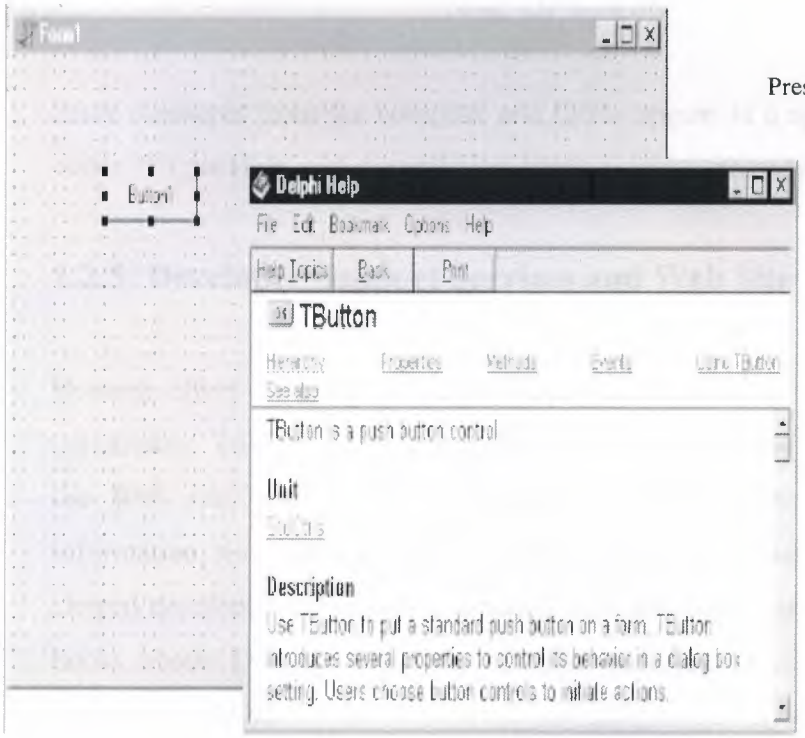
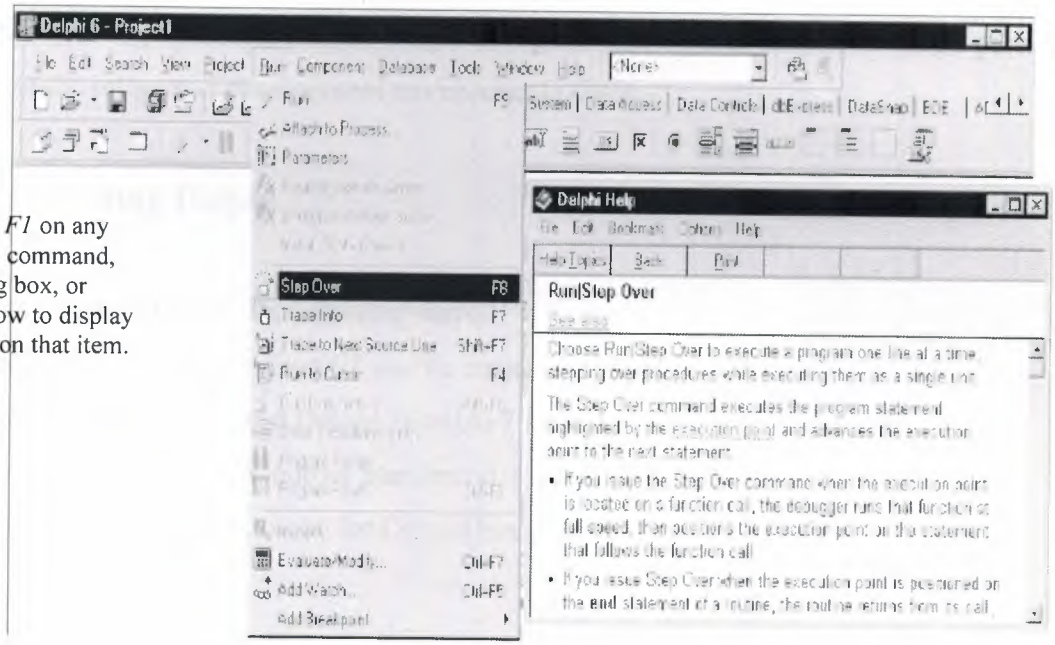


Fig. 1.5 Tbutton

Pressing the Help button in any dialog box also displays context-sensitive online documentation.



Press *F1* on any menu command, dialog box, or window to display Help on that item.

Fig. 1.6 Run/Step Over

Error messages from the compiler and linker appear in a special window below the Code editor. To get Help with compilation errors, select a message from the list and press F1.

1.2.5. 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 7|Delphi 7 from the Windows Start menu.
- Choose Run from the Windows Start menu, then enter Delphi32.
- Double-click Delphi32.exe in the Delphi\Bin directory.

1.3.2. The 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.

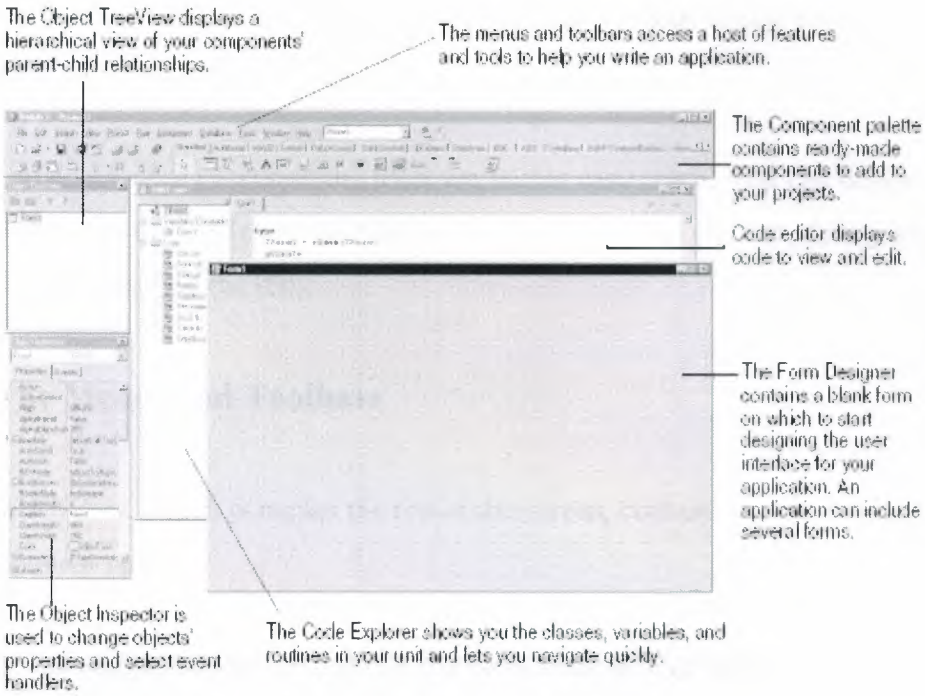


Fig. 1.7 IDE

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.

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.

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.

1.3.3. The Menus and Toolbars

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

Main window in its default arrangement

Fig. 1.8 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.

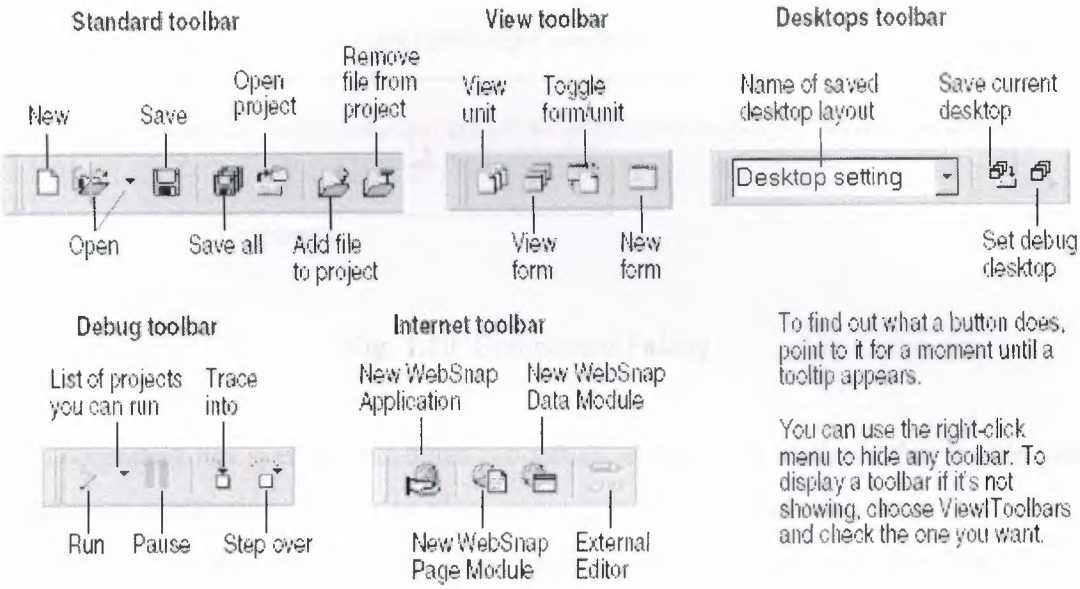


Fig. 1.9 Toolbars

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.

1.3.4. The Component Palette, Form Designer, and Object Inspector

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.

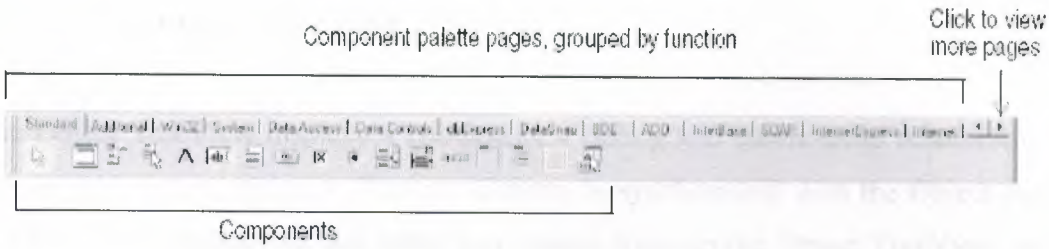


Fig. 1.10 Component Palatte

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. For the components you place on the form, use the Object Inspector to set design time properties, create event handlers, and filter visible properties and events, making the connection between your application's visual appearance and the code that makes your application run.

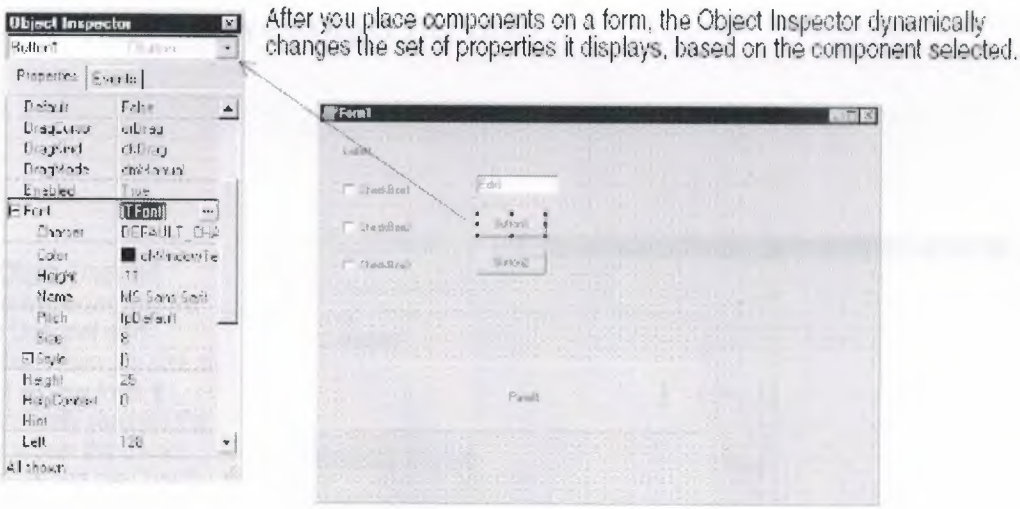


Fig. 1.11 Changing Set of Properties in Object Inspector

1.3.5. The Object TreeView

The Object TreeView 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 TreeView, both the Object Inspector and the form change focus.

You can use the Object TreeView 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 TreeView, if you drag the check box on top of the panel icon, the check box becomes the child of the panel.

If an object's properties have not been completed, the Object TreeView displays a red question mark next to it. You can also double-click any object in the tree diagram to open the Code editor to a place where you can write an event handler. If the Object TreeView isn't displayed, choose View|Object TreeView.

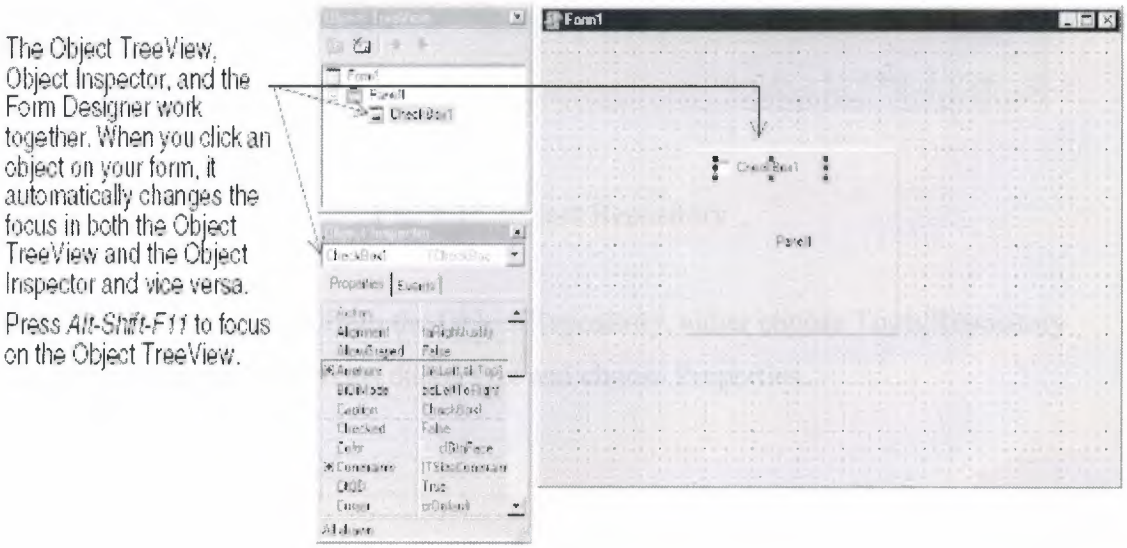


Fig. 1.12 Panel

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

1.3.6. 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. The New Items dialog box is the same as the Object Repository. Check the Repository to see if it contains an object that resembles one you want to create.

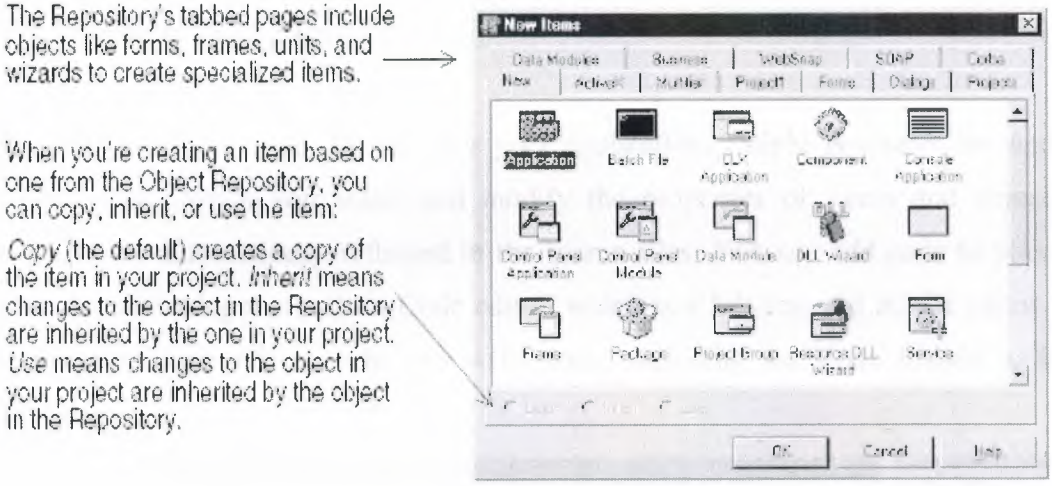


Fig. 1.13 Object Repository

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.

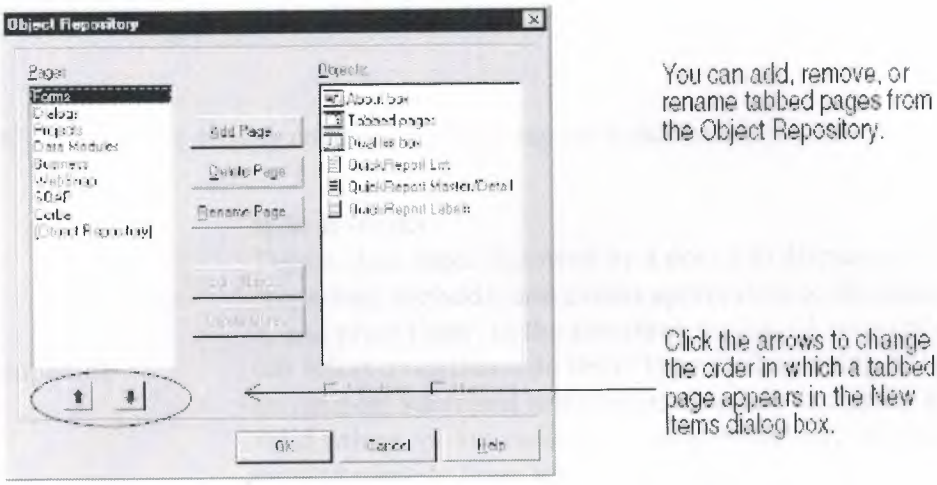


Fig. 1.14 Adding project and form templates to the Object Repository

1.3.7. 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.

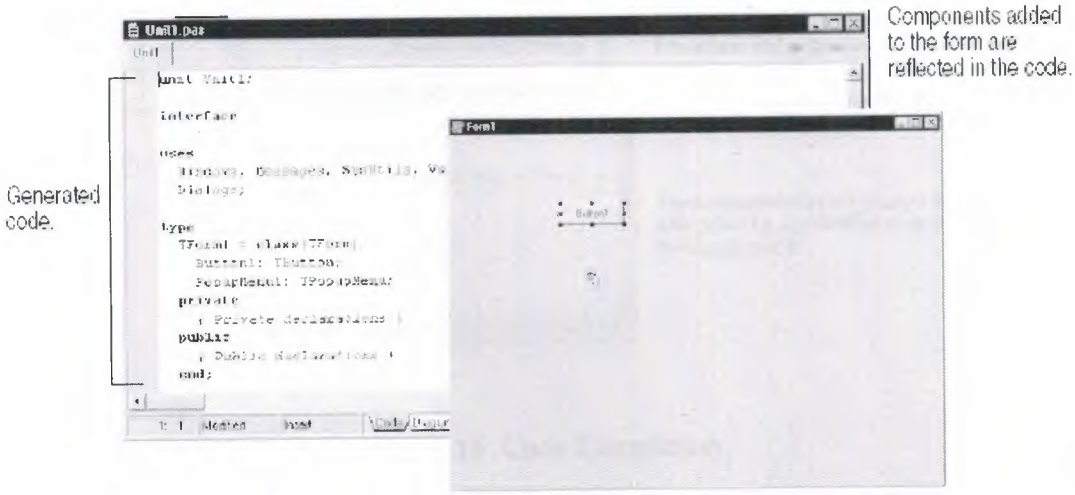


Fig. 1.15 Code Editor

1.3.7.1. Code Insight

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

Tool	How it works
Code completion	Type a class name followed by a dot (.) to display a list of properties, methods, and events appropriate to the class, select it, and press Enter. In the **interface** section of your code you can select more than one item. Type the beginning of an assignment statement and press Ctrl+space to display a list of valid values for the variable. Type a procedure, function, or method name to bring up a list of arguments.
Code parameters	Type a method name and an open parenthesis to display the syntax for the method's arguments.
Tooltip expression evaluation	While your program has paused during debugging, point to any variable to display its current value.
Tooltip symbol insight	While editing code, point to any identifier to display its declaration.
Code templates	Press Ctrl+Jto see a list of common programming statements that you can insert into your code. You can create your own templates in addition to the ones supplied with Delphi.

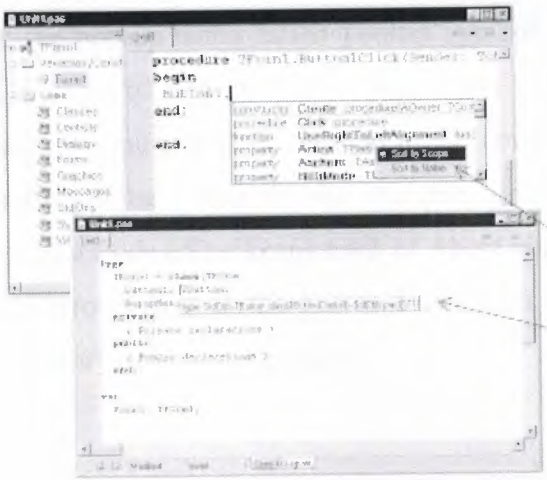


Fig. 1.16 Code Completion

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.

1.3.8. 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.

1.3.9. 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.

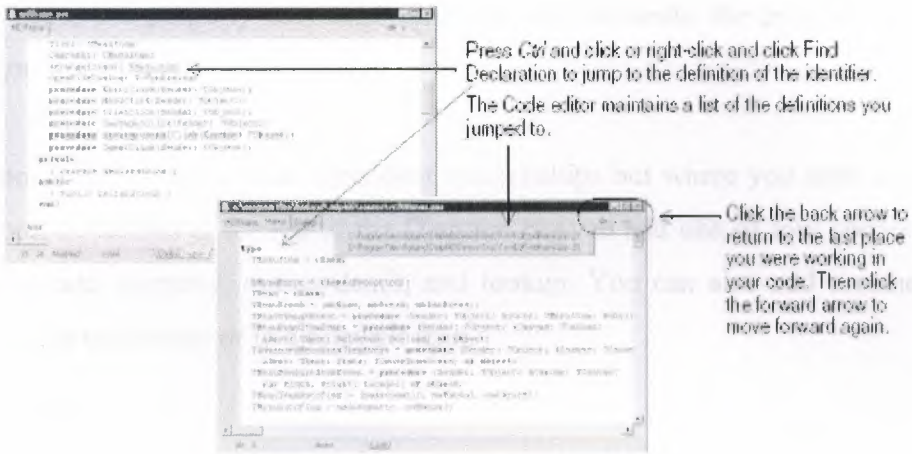


Fig. 1.17 Code Editor

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".

1.3.10. 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. From the Object TreeView, simply drag one or multiple icons to the Diagram page to arrange them vertically. To arrange them horizontally, press Shift while dragging. When you drag icons with parentchildren or component-property dependencies onto the page, the lines, or connectors, that display the dependent relationships are automatically added. For example, if you add a dataset component to a data module and drag the dataset icon plus its property icons to the

Diagram page, the property connector automatically connects the property icons to the dataset icon.

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.

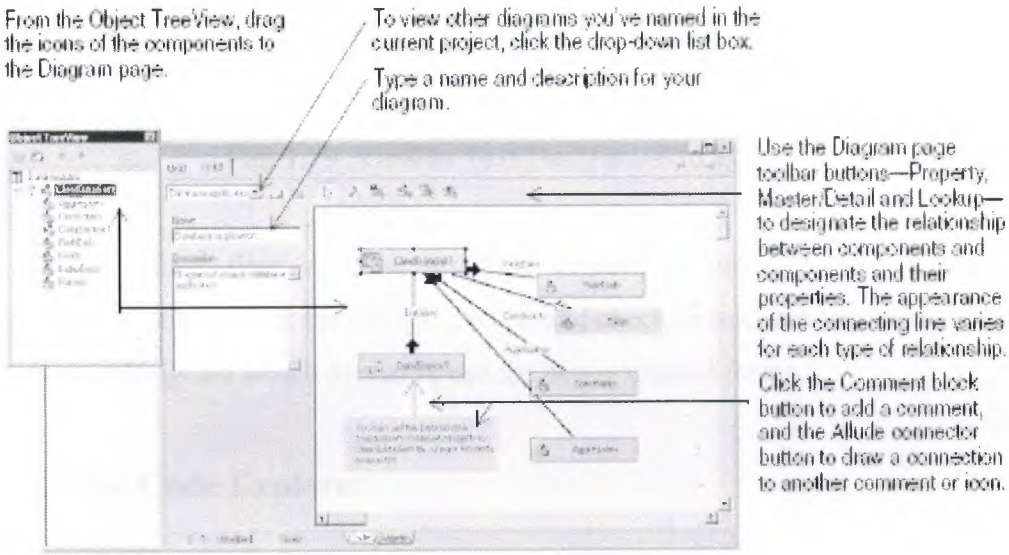


Fig. 1.18 Diagram Page Toolbar Button

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

1.3.11. Viewing 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.

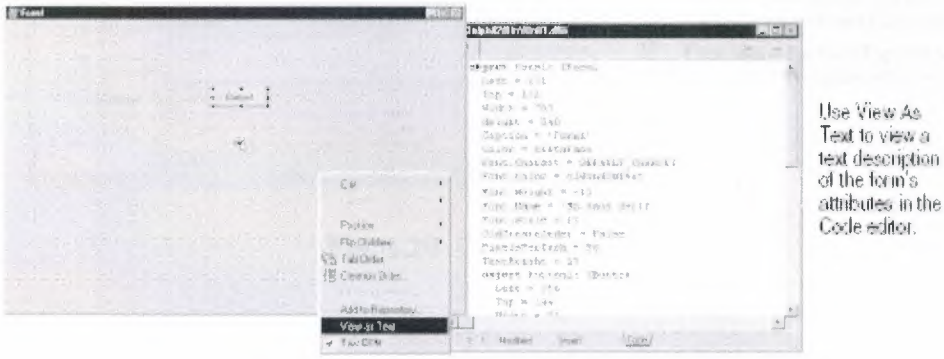


Fig. 1.19 View as Text Description of Form

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.

1.3.12. 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.

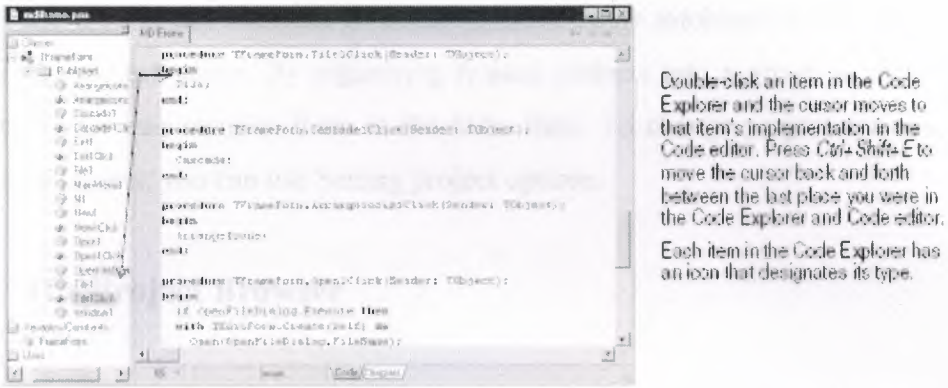


Fig. 1.20 Code Explorer

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

1.3.13. 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.

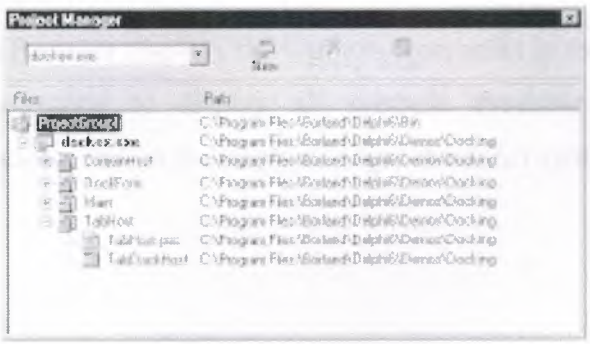
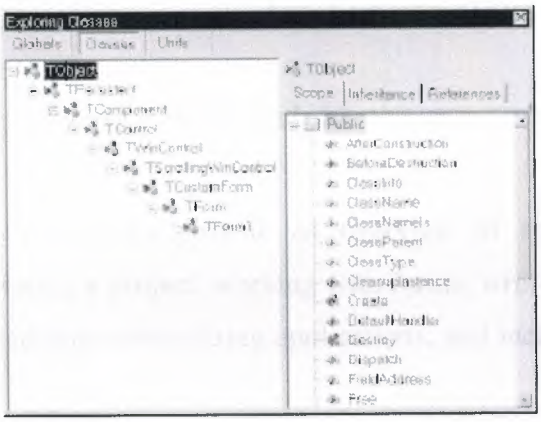


Fig. 1.21 Project Manager

You can use the Project Manager to combine and display information on related projects into a single project group. By organizing related projects into a group, such as multiple executables, you can compile them at the same time. To change project options, such as compiling a project, you can use Setting project options.

1.3.14. 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.



The Project Browser has two resizeable panes: the Inspector pane (on the left) and the Details pane. The Inspector pane has three tabs for globals, classes, and units.

Globals displays classes, types, properties, methods, variables, and routines.

Classes displays classes in a hierarchical diagram.

Units displays units, identifiers declared in each unit, and the other units that use and are used by each unit.

Fig. 1.22 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.

1.3.15. To-do lists

To-do lists record items that need to be completed for a project. You can add ojectwide items to a list by adding them directly to the list, or you can add specific items directly in

the source code. Choose View|To-Do List to add or view information associated with a project.

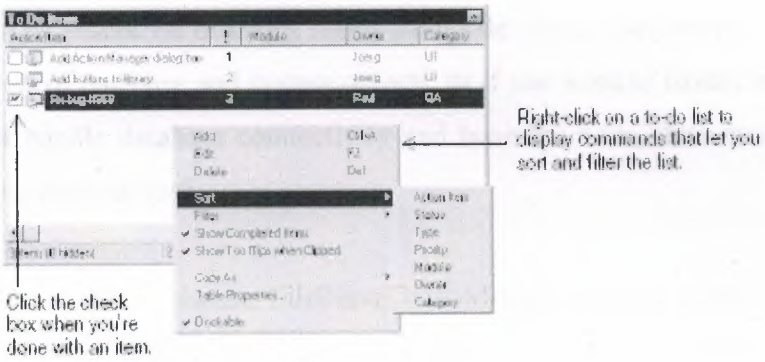


Fig. 1.23 To-Do List

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. File|New| Other opens the Object Repository, which provides additional forms, modules, and frames as well as predesigned templates such as dialog boxes to add to your project. When you start a project, you have to know what you want to develop, such as an application or DLL.

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. Delphi opens an empty data module, which displays an additional unit file for the module in the Code editor, and adds the module to the current project as a new unit. Add nonvisual components to a data module in the same way as you would to a form.

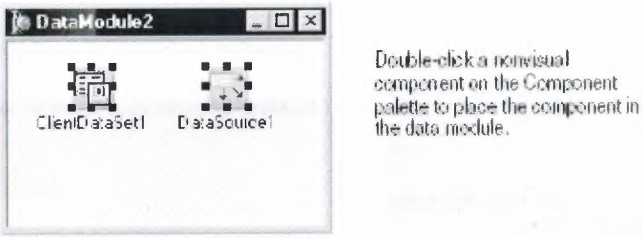


Fig. 1.24 Adding Data Modules

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

To place components on a form, either:

1 Double-click the component; or

2 Click the component once and then click the form where you want the component to appear.

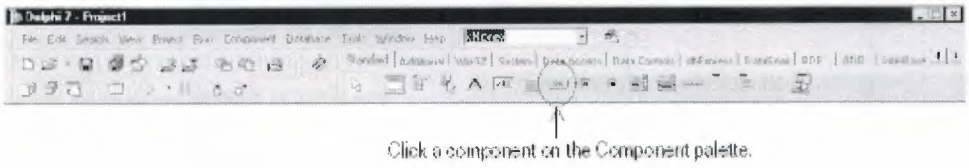


Fig. 1.25 Component Palette

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

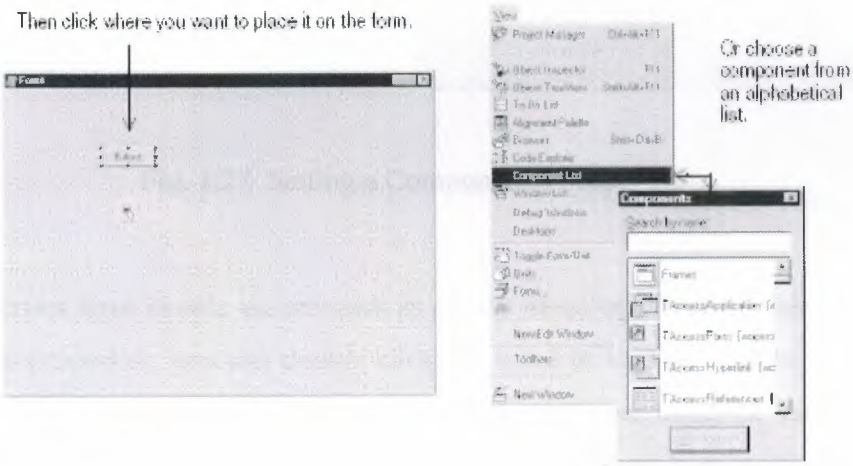


Fig. 1.26 Component List

1.4.5. Setting Component Properties

After you place components on a form, set their properties and code their event handlers. Setting a component's properties changes the way a component appears and behaves in

your application. When a component is selected on a form, its properties and events are displayed in the Object Inspector.

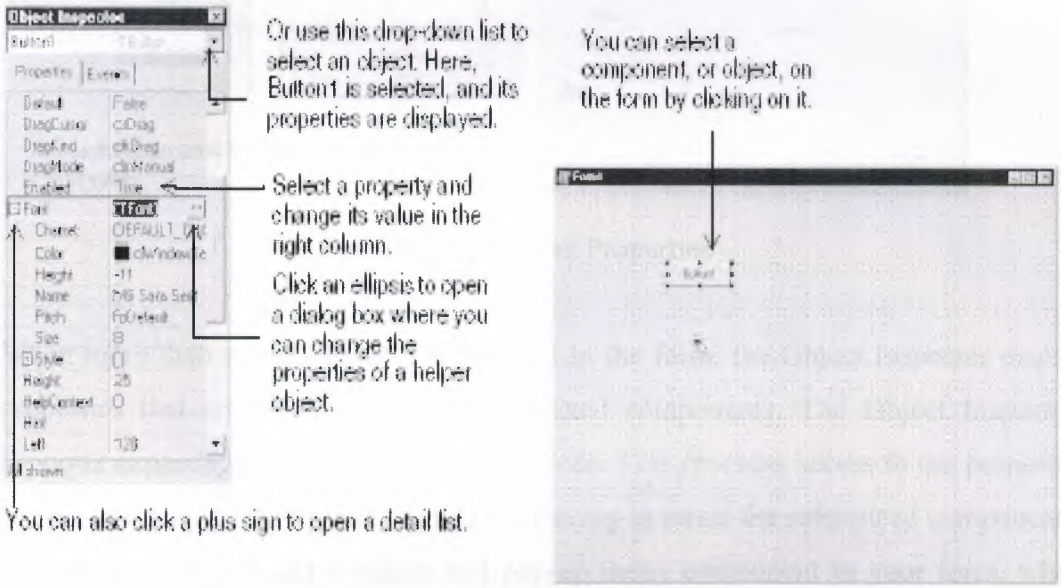


Fig. 1.27 Setting a Component's Properties

Many properties have simple values such as names of colors, True or False, and integers. For Boolean properties, you can double click the word to toggle between True and False. Some properties have associated property editors to set more complex values. When you click on such a property value, you'll see an ellipsis. For some properties, such as size, enter a value.

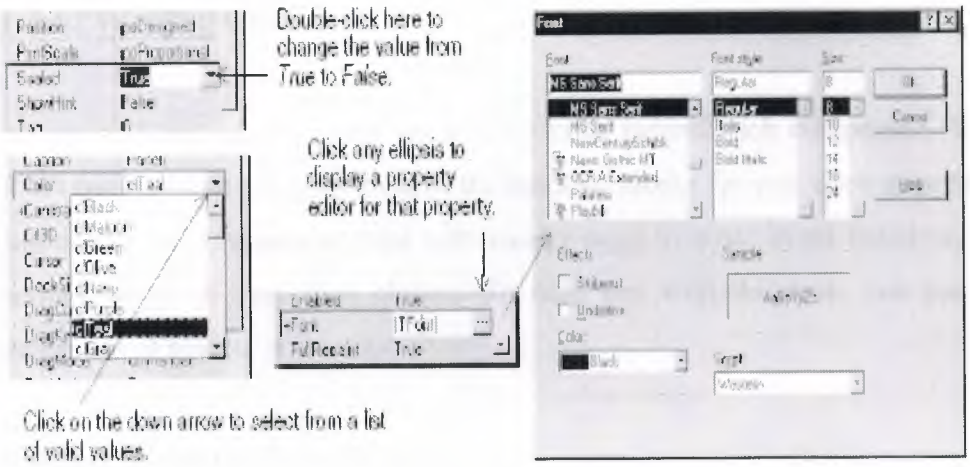


Fig. 1.28 Font Properties

When more than one component is selected in the form, the Object Inspector displays all properties that are shared among the selected components. The Object Inspector also supports expanded inline component references. This provides access to the properties and events of a referenced component without having to select the referenced component itself. For example, if you add a button and pop-up menu component to your form, when you select the button component, in the Object Inspector you can set the PopupMenu property to PopupMenu1, which displays all of the pop-up menu's properties.

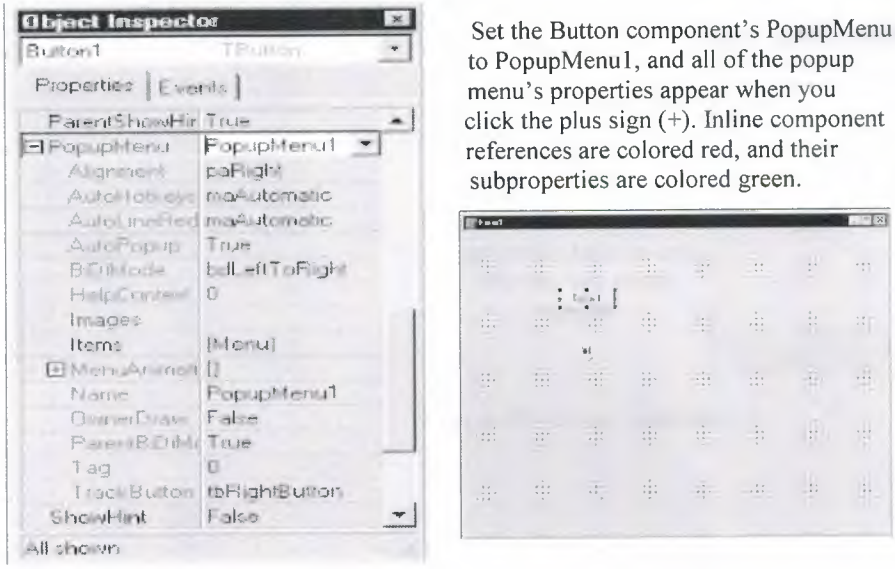


Fig. 1.29 Setting the Button Component's Popup Menu

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.

1.4.6.1. Writing Event Handlers

Your code may need to respond to events that might occur to a component at runtime. An event is a link between an occurrence in the system, such as clicking a button, and a piece of code that responds to that occurrence. The responding code is an event handler. This code modifies property values and calls methods. To view predefined event handlers for a component on your form, select the component and, on the Object Inspector, click the Events tab.

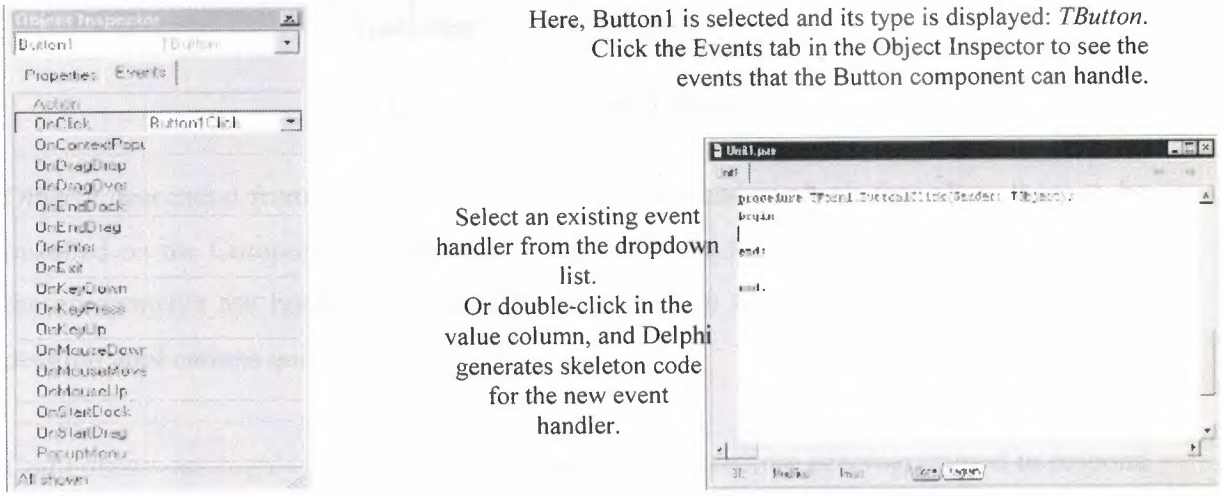


Fig. 1.30 Event Handlers

1.4.6.2. 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. The following diagram shows some of the principal classes that make up the VCL hierarchy. The CLX hierarchy is similar.

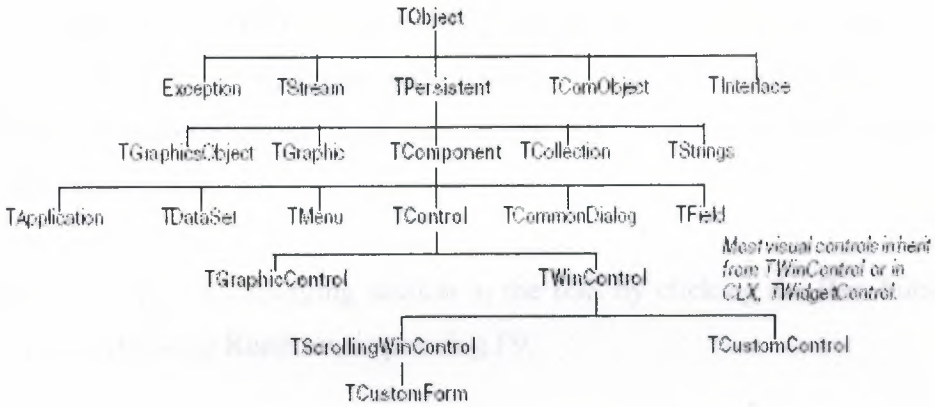


Fig. 1.31 Component Library

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. Most editions of Delphi come with the component library source code and examples of Delphi programming techniques.

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.

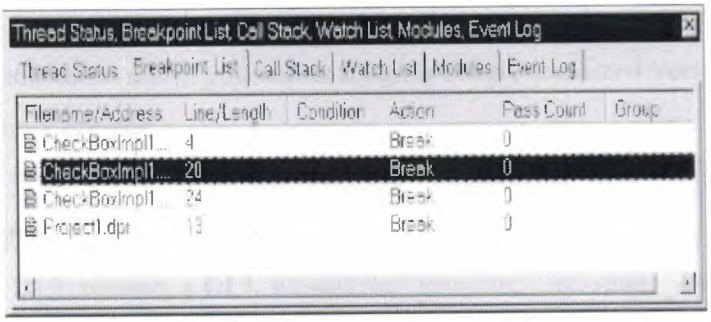
Delphi uses an integrated debugger so that you can control program execution, watch variables, and modify data values. You can step through your code line by line, examining the state of the program at each breakpoint. To use the integrated debugger, choose Tools|Debugger Options, click the General page, 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.

Fig. 1.32 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.



You can combine several debugging windows for easier use.

Fig. 1.33 Debugging

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.

1.4.8. 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.

1.4.9. 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.

1.4.10. 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.

1.4.10.1. 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.

1.4.10.2. 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 dbExpress 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.

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

1.4.11. BDE Administrator

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.

1.4.12. SQL Explorer (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.

1.4.13. Database Desktop

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

1.4.14. 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.

1.4.15. Custom Components

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.

1.4.16. DLLs

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.

1.4.17. 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.

1.4.18. 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. Customizing The Desktop

This chapter explains some of the ways you can customize the tools in Delphi's IDE.

1.5.1. Organizing Your Work Area

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.

1.5.2. 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.

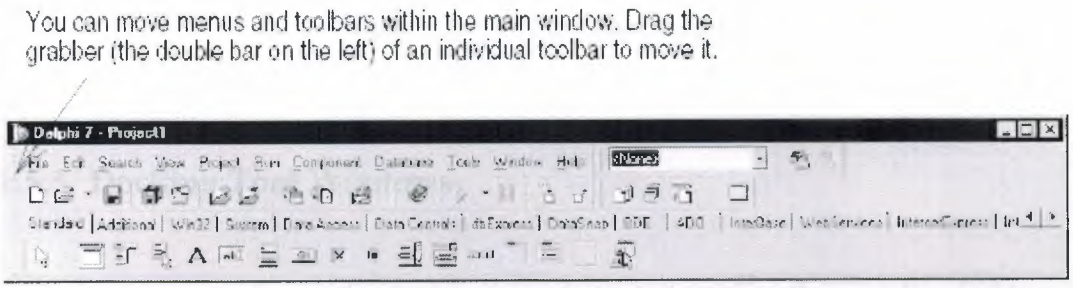
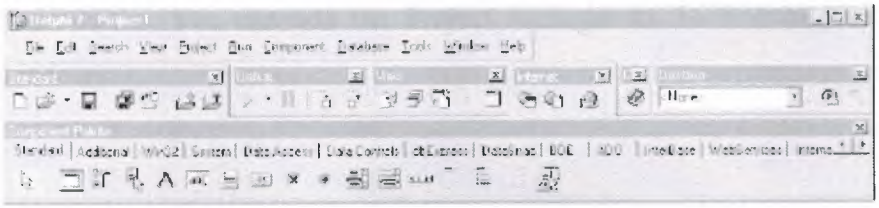


Fig. 1.34 Arranging Menus and Toolbars

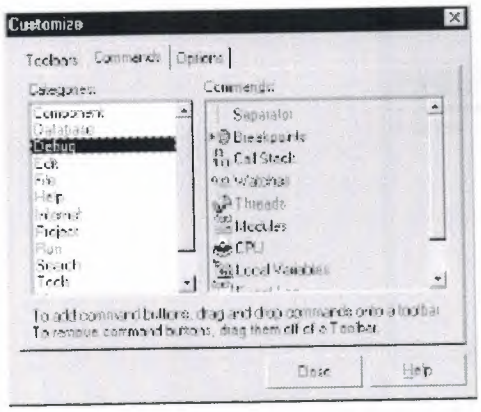
You can separate parts from the main window and place them elsewhere on the screen or remove them from the desktop altogether. This is useful if you have a dual monitor setup.



Main window organized differently.

Fig. 1.35 Main Window

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.



On the Commands page, select any command and drag it onto any toolbar.

On the Options page, click Show tooltips to make sure the hints for components and toolbar icons appear.

Fig. 1.36 Customize Command

1.5.3. Docking 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.

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.

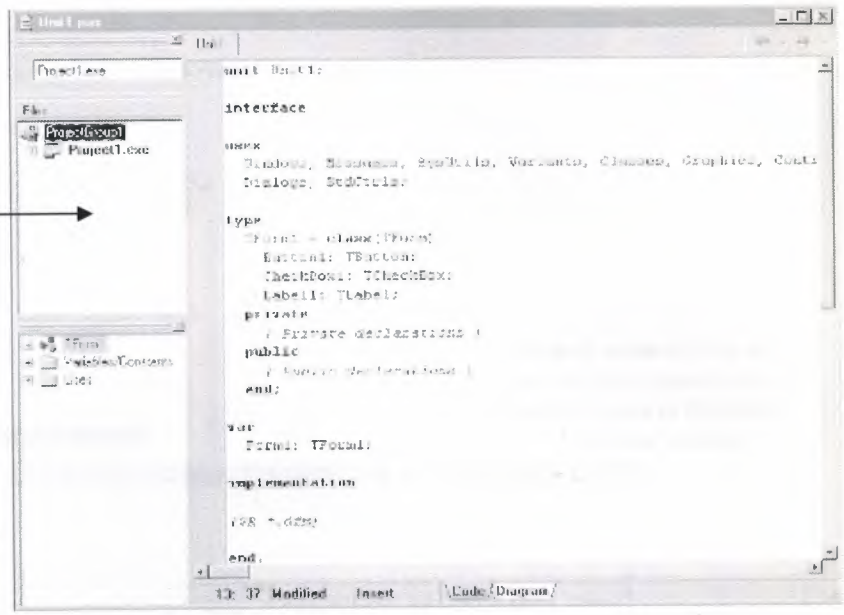


Fig. 1.37 Docking Tool Windows

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.

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

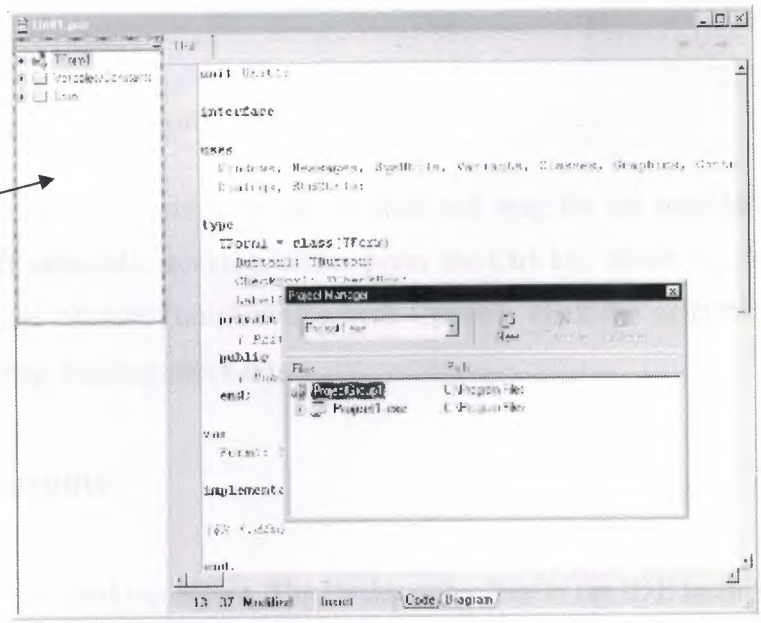
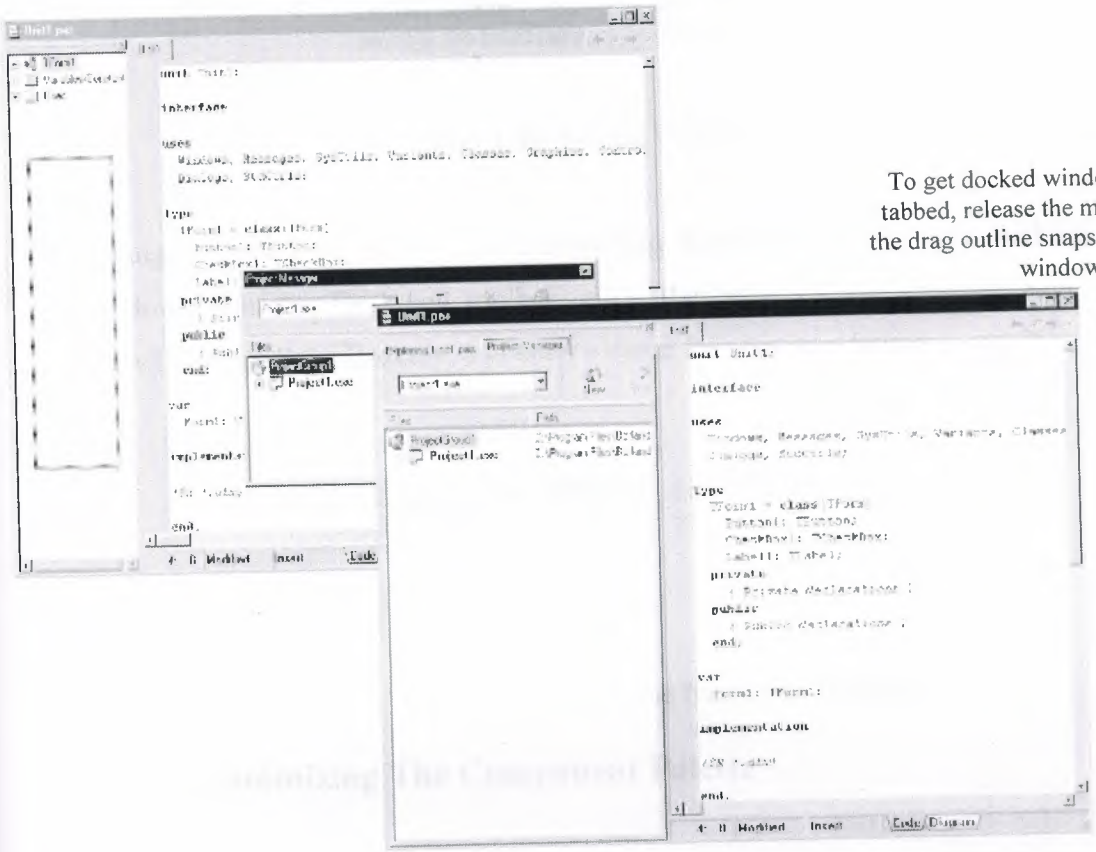


Fig. 1.38 Two Windows Snap Together

You can also dock tools to form tabbed windows.



To get docked windows that are tabbed, release the mouse *before* the drag outline snaps to the other window's corner.

Fig. 1.39 Docking Tools to Form

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.

1.5.4. Saving 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.

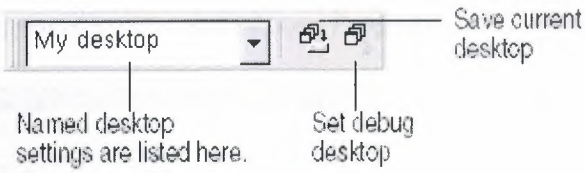


Fig. 1.40 Saving Desktop Layouts

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.

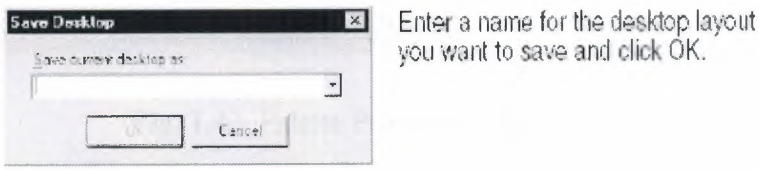


Fig. 1.41 Entering the Name For Desktop

1.5.5. Customizing The Component Palette

In its default configuration, the Component palette displays many useful objects organized functionally onto tabbed pages. You can customize the Component palette by:

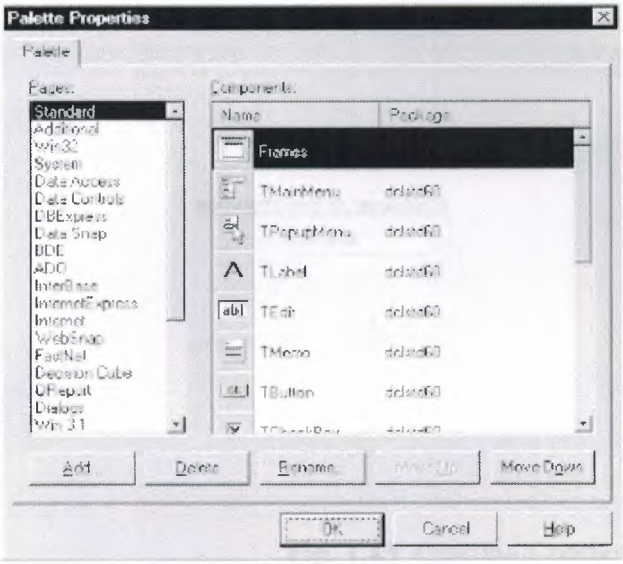
- Hiding or rearranging components.
- Adding, removing, rearranging, or renaming pages.
- Creating component templates and adding them to the palette.
- Installing new components.

1.5.6. Arranging 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.

Fig. 1.42 Palette Properties Dialog Box

1.5.7. 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.

After placing a template on a form, you can reposition the components independently, reset their properties, and create or modify event handlers for them just as if you had

placed each component in a separate operation.

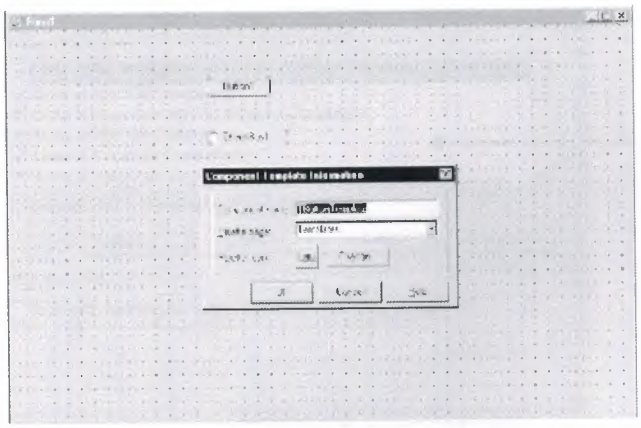
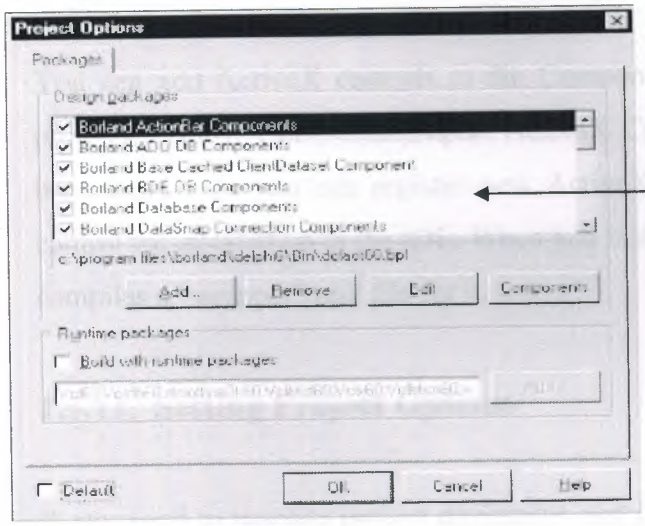


Fig. 1.43 Creating Component Templates

1.5.8. Installing Component Packages

Whether you write custom components or obtain them from a vendor, the components must be compiled into a package before you can install them on the Component palette.A package is a special DLL containing code that can be shared among Delphi applications, the IDE, or both. Runtime packages provide functionality when a user runs an application. Design-time packages are used to install components in the IDE.

Delphi packages have a .bpl extension. If a third-party vendor's components are already compiled into a package, either follow the vendor's instructions or choose Component|Install Packages.



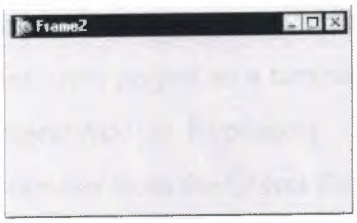
These components come preinstalled in Delphi. When you install new components from third-party vendors, their package appears in this list. Click Components to see what

Fig. 1.44 Installing Component Packages

1.5.9. Using Frames

A frame (TFrame), like a form, is a container for components that you want to reuse. A frame is more like a customized component than a form. Frames can be saved on the Component palette for easy reuse and they can be nested within forms, other frames, or other container objects. After a frame is created and saved, it continues to function as a unit and to inherit changes from the components (including other frames) it contains. When a frame is embedded in another frame or form, it continues to inherit changes made to the frame from which it derives.

To open a new frame, choose File | New | Frame.



You can add whatever visual or nonvisual components you need to the frame. A new unit is automatically added to the Code editor.

Fig. 1.45 Opening New Frame

1.5.10. Adding ActiveX Controls

You can add ActiveX controls to the Component palette and use them in your Delphi projects. Choose Component|Import ActiveX Control to open the Import ActiveX dialog box. From here you can register new ActiveX controls or select an already registered control for installation in the IDE. When you install an ActiveX control, Delphi creates and compiles a "wrapper" unit file for it.

1.5.11. Setting Project Options

If you need to manage project directories and to specify form, application, compiler, and linker options for your project, choose Project |Options. When you make changes in the Project Options dialog box, your changes affect only the current project; but you can also save your selections as the default settings for new projects.

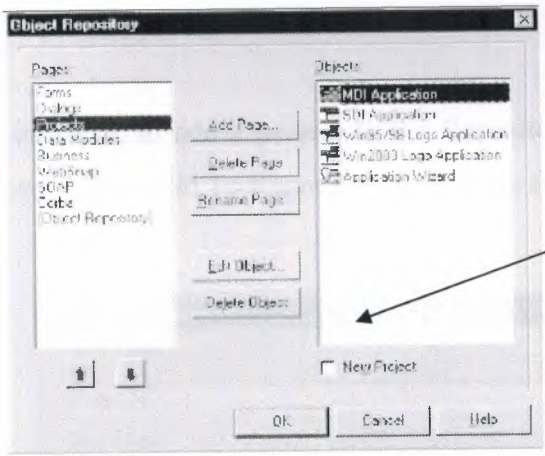
1.5.12. Setting Default Project Options

To save your selections as the default settings for all new projects, in the lower left corner of the Project Options dialog box, check Default. Checking Default writes the current settings from the dialog box to the options file defproj.dof, located in the Delphi7\Bin directory. To restore Delphi's original default settings, delete or rename the defproj.dof file.

1.5.13. Specifying Project and Form Templates As The Default

When you choose File|New|Application, Delphi creates a standard new application with an empty form, unless you specify a project template as your default project. You can save your own project as a template in the Object Repository on the Projects page by choosing Project|Add to Repository . Or you can choose from one of Delphi's existing project templates from the Object Repository.

To specify a project template as the default, choose Tools or Repository. In the Object Repository dialog box, under Pages, select Projects. If you've saved a project as a template on the Projects page, it appears in the Objects list. Select the template name, check New Project, and click OK.



The Object Repository's pages contain project templates only, form templates only, or a combination of both. To set a project template as the default, select an item in the Objects list and check New Project .To set a form template as the default, select an item in the Objects list and check New Form or Main Form.

Fig. 1.46 Specifying Project and Form Templates As The Default

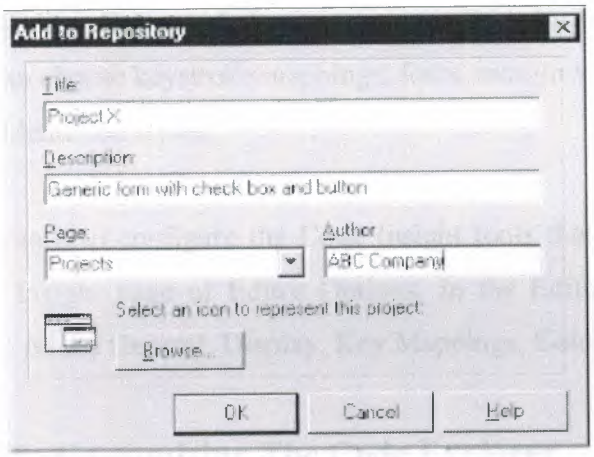
Once you've specified a project template as the default, Delphi opens it automatically whenever you choose File |New |Application.

In the same way that you specify a default project, you can specify a default new form and a default main form from a list of existing form templates in the Object Repository. The default new form is the form created when you choose File |New |Form to add an additional form to an open project. The default main form is the form created when you open a new application. If you haven't specified a default form, Delphi uses a blank form. You can override your default project or form temporarily by choosing File |New| Other and selecting a different template from the New Items dialog box.

1.5.14. Adding Templates To The Object Repository

You can add your own objects to the Object Repository as templates to reuse and share with other developers over a network. Reusing objects lets you build families of applications with common user interfaces and functionality that reduces development time and improves quality.

For example, to add a project to the Repository as a template, first save the project and choose Project|Add To Repository. Complete the Add to Repository dialog box.



Enter a title, description, and author. In the Page list box, choose Projects so that your project will appear on the Repository's Projects tabbed page.

Fig. 1.47 Adding Templates To The Object Repository

The next time you open the New Items dialog box, your project template will appear on the Projects page (or the page to which you had saved it).

1.5.15. Setting Tool Preferences

You can control many aspects of the appearance and behavior of the IDE, such as the Form Designer, Object Inspector, and Code Explorer. These settings affect not just the current project, but projects that you open and compile later. To change global IDE settings for all projects, choose Tools |Environment Options.

1.5.16. Customizing The Form Designer

The Designer page of the Tools|Environment Options dialog box has settings that affect the Form Designer. For example, you can enable or disable the "snap to grid" feature, which aligns components with the nearest grid line; you can also display or hide the names, or captions, of nonvisual components you place on your form.

1.5.17. Customizing The Code Editor

One tool you may want to customize right away is the Code editor. Several pages in the Tools|Editor Options dialog box have settings for how you edit your code. For example, you can choose keystroke mappings, fonts, margin widths, colors, syntax highlighting, tabs, and indentation styles.

You can also configure the Code Insight tools that you can use within the editor on the Code Insight page of Editor Options. In the Editor Options dialog box, click the Help button on the General, Display, Key Mappings, Color, and Code Insight pages.

1.5.18. Customizing The Code Explorer

When you start Delphi, the Code Explorer opens automatically. If you don't want Code Explorer to open automatically, choose Tools|Environment Options, click the Explorer tab, and uncheck Automatically show Explorer.

You can change the way the Code Explorer's contents are grouped within the Code Explorer by right-clicking in the Code Explorer, choosing Properties, and, under Explorer categories, checking and unchecking the check boxes. If a category is checked, elements in that category are grouped under a single node. If a category is unchecked, each element in that category is displayed independently on the diagram's trunk. For example, if you uncheck the Published category, the Published folder disappears but not the items in it.

In the Code Explorer, you can sort all source elements alphabetically or in the order in which they are declared in the source file

To display the folder for each type of source element in the Code Explorer, check an Explorer category.

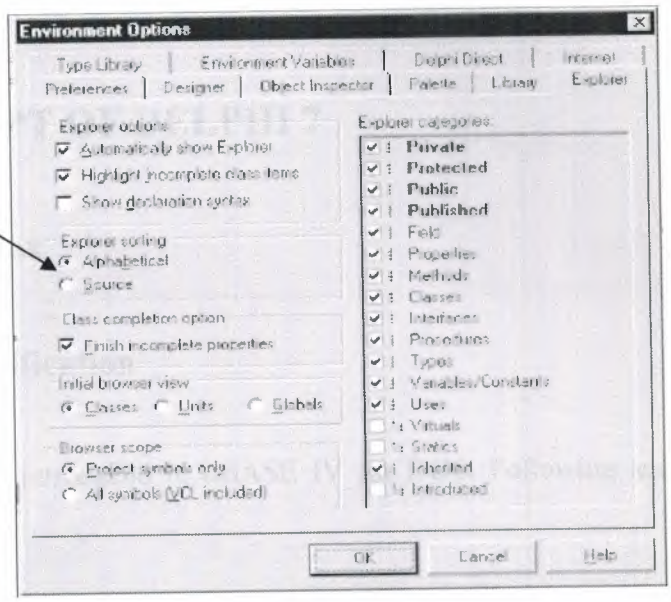


Fig. 1.48 Customizing The Code Explorer

CHAPTER 2

2. DATABASE CONCEPT OF DELPHI 7

2.1 About Dbase And Paradox

2.1.1 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.

2.1.2. 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.

2.1.3. 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).

2.2. Paradox Standard Table Specifications

Also known as 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.

2.2.1. 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.

2.2.2. 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

2.2.2.1. Paradox Field Types

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 3

3. DATABASE DESIGN OF THE PROGRAM

3.1 Database Design of The Program

The stock database consists of seven tables those are stock list table, company table, customer table, personel table, order table, sale table, login table.

Company table contains six fields :

- Company Id
- Name
- Phone
- Fax
- Address
- Web

Stock List table contains nine fields :

- Stock Id
- Stock Code
- Stock Name
- Purchase Price
- Selling Price
- Company
- Guarantee
- Date
- Quantity

Customer table contains seven fields :

- Customer Id
- Name
- Surname
- Fax
- Phone
- E-mail
- Address

Customer table contains ten fields :

- Personel No
- First Name
- Last Name
- Duty
- Department
- Salary
- Address
- Phone
- E-mail
- Hire Date

Sale table contains six fields :

- Sale Id
- Customer Id
- Product Id
- Date
- Price
- Quantity

Order table contains eight fields :

- Order Id
- Order Name
- Properties
- Quantity
- Company
- Price
- Order Date
- Arrival Date

Login table contains threefields :

- User Id
- User Name
- User Password

<u>The relationships between tables will as follows:</u>

In Company Table Company Id field is a primary key.

In Order Table Order Id is a primary key.

In Stock List Table Stock Id is a primary key.

In Customer Table Customer Id is a primary key.

In Sale Table Sale Id is a primary key.

In Login Table User Id is a primary key.

In Personel No is a primary key.

When you execute the program Login Form opens, then it will ask you username and password. You can see it in Fig 3.1.

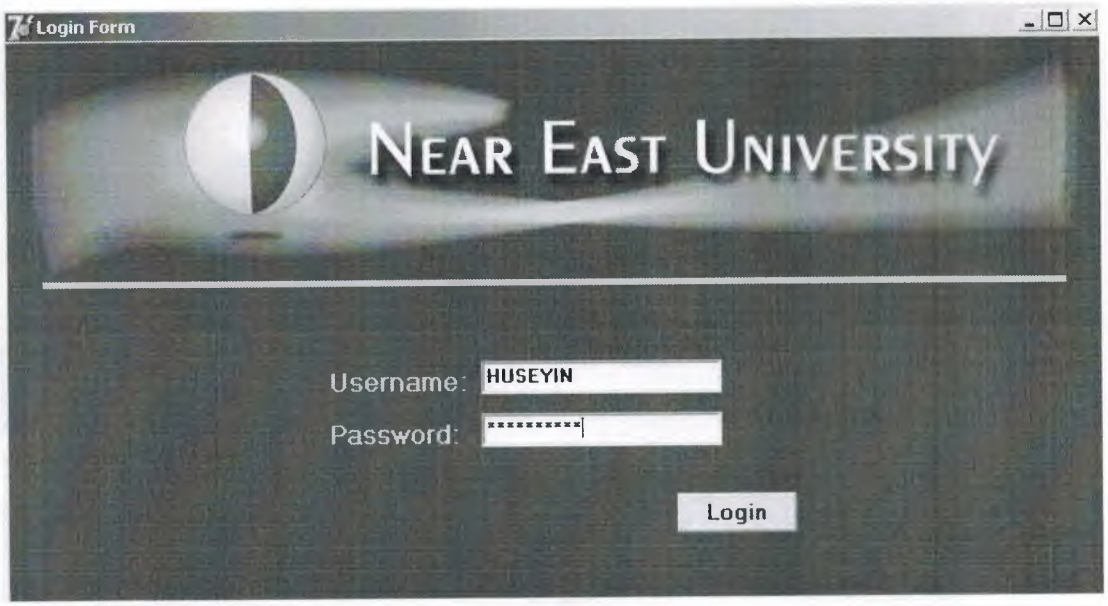


Fig 3.1 Login Form

If you do not know username and password you can not login this program. In fig3.1 write username and password then Click Login Button or Press Enter Key to login this program.

In this program there is Stock , Customer, Personnel, Order , Companies , Report , Users , Calendar , About and Exit menus (shown in Fig 3.2)

Fig 3.2 Main Form

In Stock Menu there are two submenus. These are Product Entry Submenu and Stock List Submenu.

Fig 3.3 Product Entry Submenu

When you select Product Entry Submenu then Product Entry form is appeared.

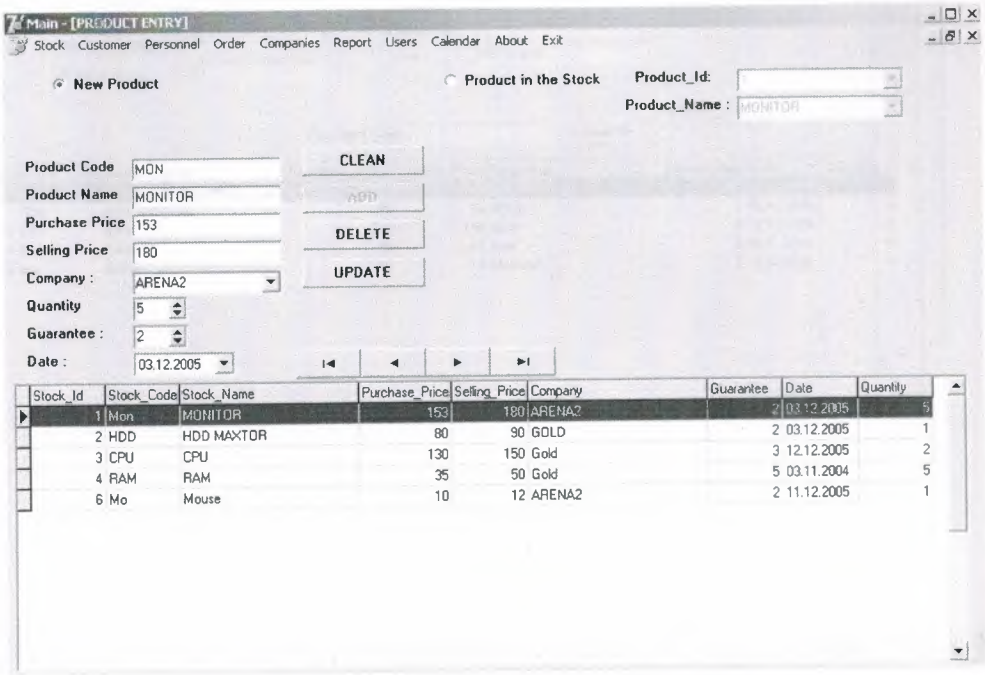


Fig 3.4 Product Entry Form

In this form we can add new product to the database then it is shown in the Fig 3.4 on the form. And also we can delete the product which was added before after select on the table. Moreover we can delete the product which we have added before. And when we press the clean button it clears all the texts. Also we can go next and prior by pressing next and previous button of the navigator. If the product exists in the stock, then we can select "Product in the stock" radio button and select in the table as well, then we can update its details as well.

When you select Stock List Submenu then Stock List form is appeared.

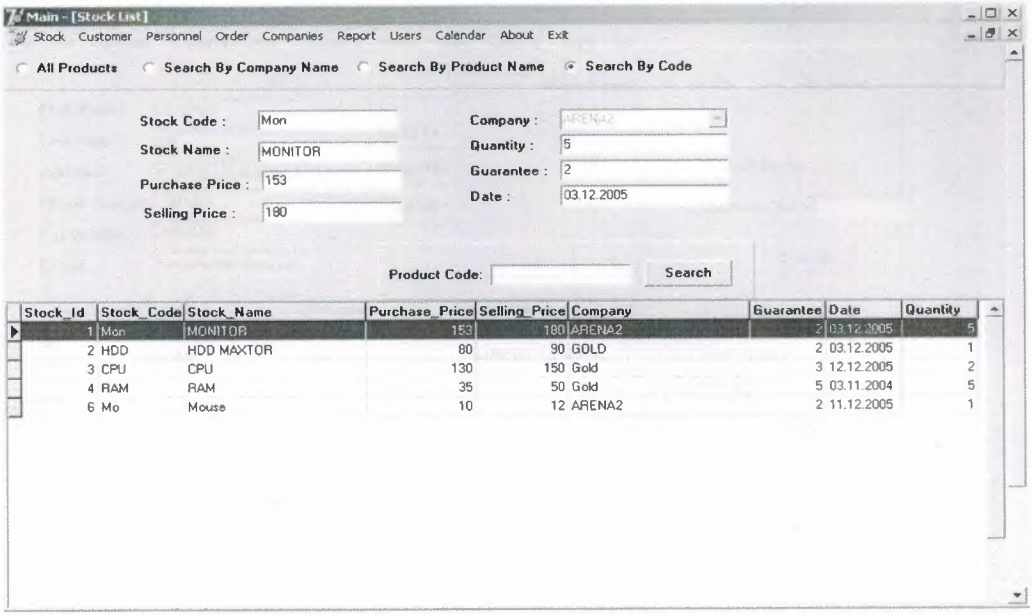


Fig 3.5 Stock List

In this form we can see all the product details by Stock_Id , Stock_Code , Stock_Name , Purchase_Price , Selling_Price , Company , Guarantee , Date and Quantity. And we can search the product by Company Name , search by Product Name and search by Product Code as well.

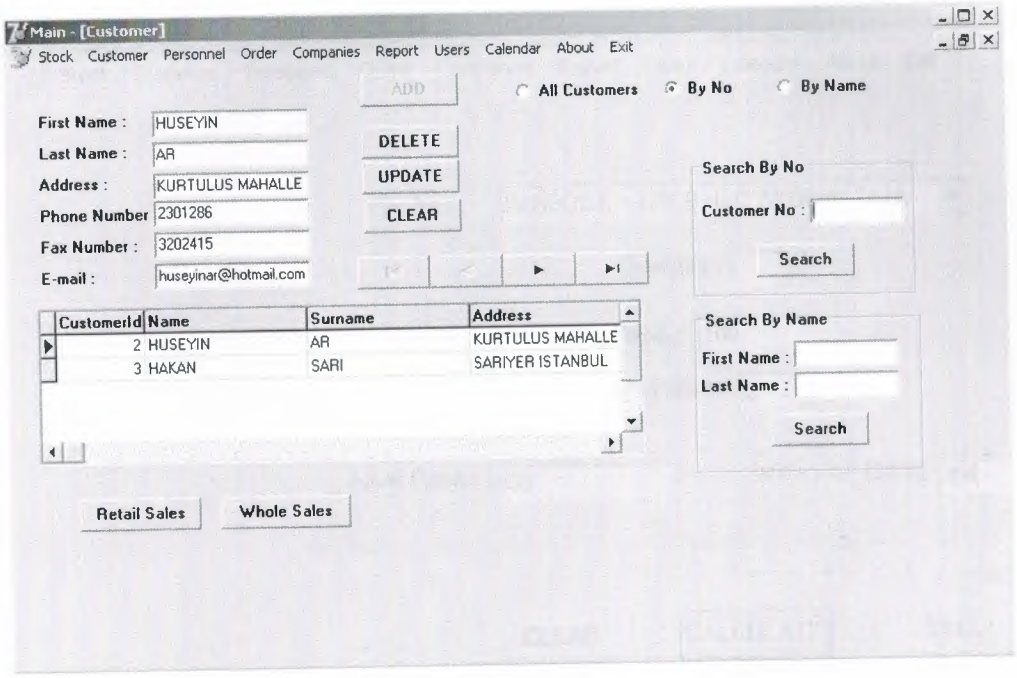


Fig 3.6 Customer Form

Here Customer details are held in this form. We can add customer, delete customer, update customer in this form. Moreover we can search customer by Id and by First Name and Surname in the same form. Also there is sale button in the same form which are Retail Sale and Whole Sale. When you select one customer then click one of the sale button the sale form appears. To sell retail product you should click the "Retail Sale" button and to sell wholesale product should click the "Wholesale" button. After that there will appear sale form to do sale process.

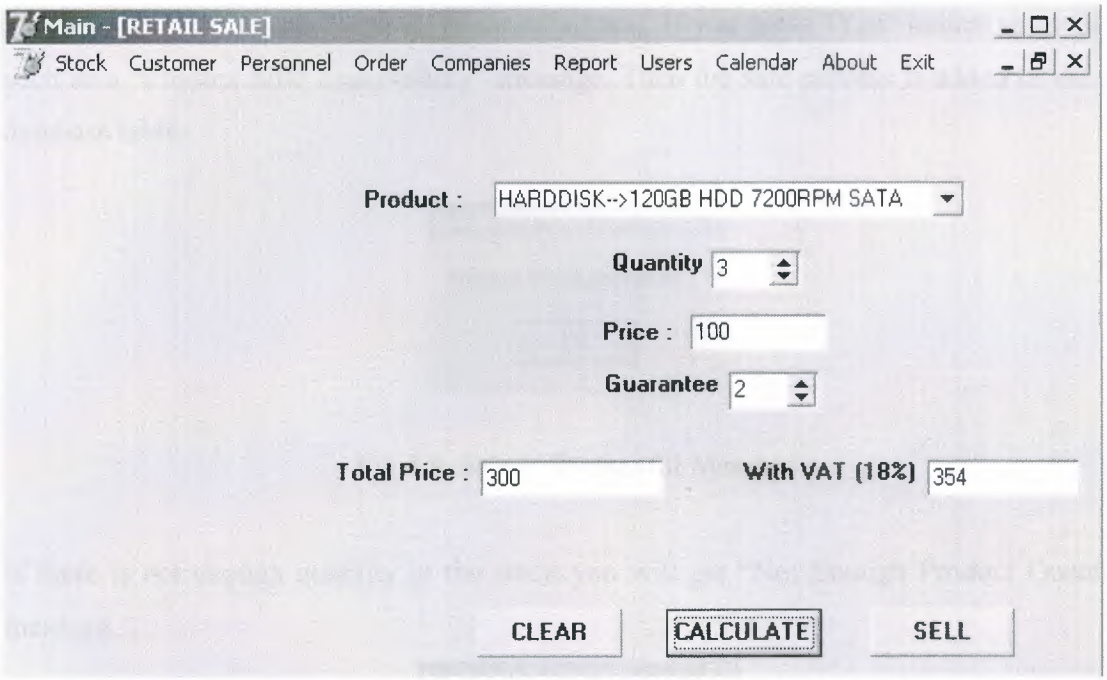


Fig 3.7 Retail Sale Form

In retail sale form the customer select one of the products which is shown in ComboBox, then customer determines the quantity. After that the price and guarantee are appeared automatically. Then when it is clicked the "Calculate" button then "Total Price" and "With VAT (18%)" are calculated and displayed on the form. When you press the "SELL" button you will see such as a "Please Confirm Sale" message:

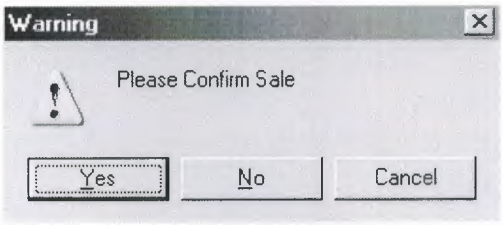


Fig 3.8 Confirm Sale Message

If you press "No" button the sale is cancelled and, If you press "Yes" button you will get such as a "Product Sold Successfully" message. Then the sale process is added to the sale database table.

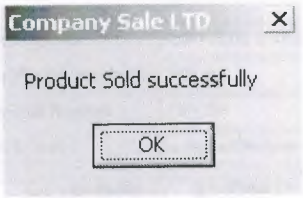


Fig 3.9 Sale is Successful Message

If there is not enough quantity in the stock you will get "Not Enough Product Quantity" message.

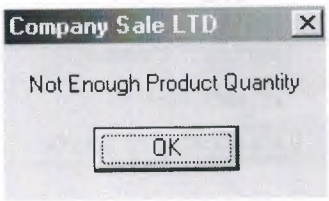


Fig 3.10 Quantity Message

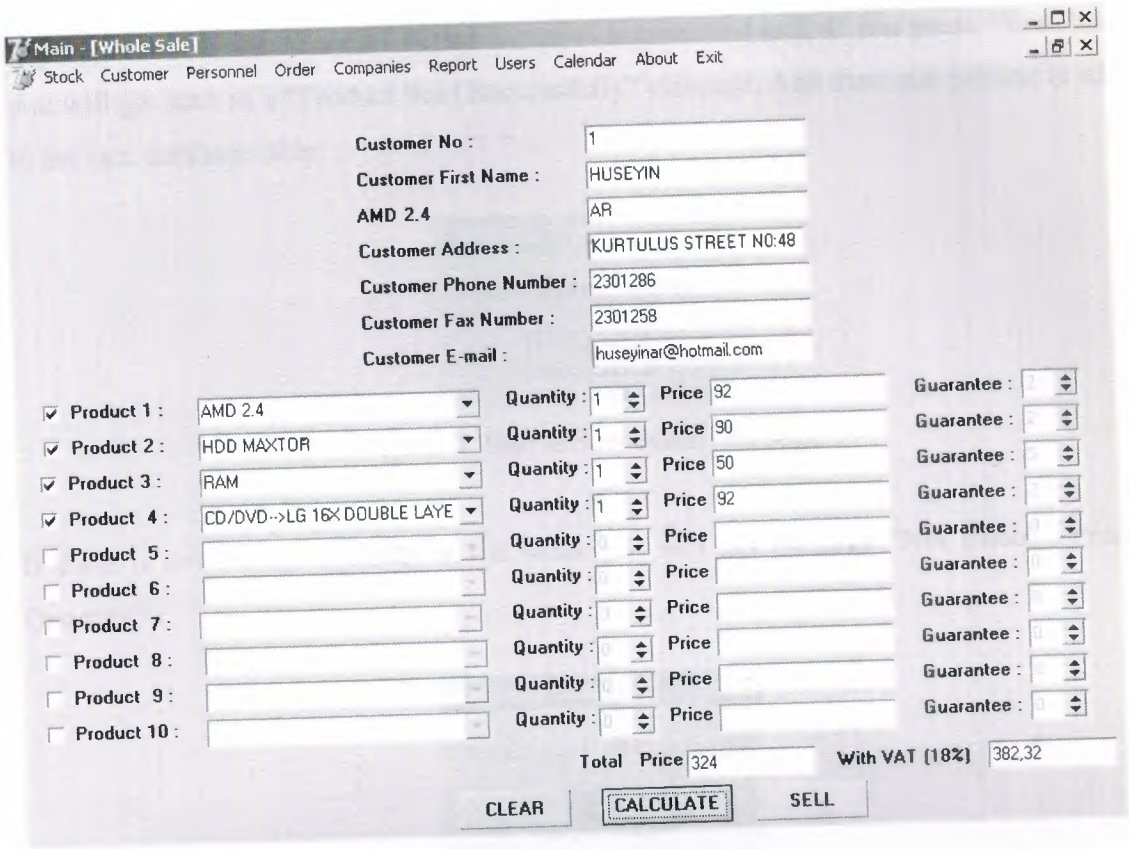


Fig 3.10 Wholesale Form

In Wholesale form at the first clicked checkboxes then Comboboxes, Quantities, Prices, Guarantees will be activated. Then we can select the products in the Comboboxes and determine the quantity. Then when it is clicked the "CALCULATE" button "Total Price" and "With VAT (18%)" are calculated and displayed on the form. When you press the "SELL" button you will see such as a "Please Confirm Sale" message:

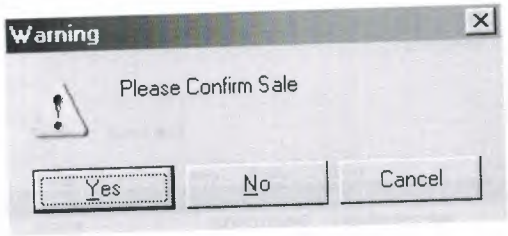


Fig 3.11 Confirm Sale Message

If you press "No" or "Cancel" button the sale is cancelled and, If you press "Yes" button you will get such as a "Product Sold Successfully" message. And then sale process is added to the sale database table.

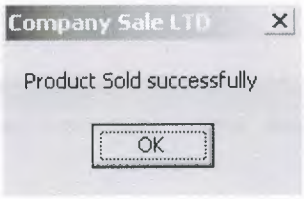


Fig 3.12 Sale is Successful Message

If there is not enough quantity in the stock you will get message "Not Enough Product Quantity" .

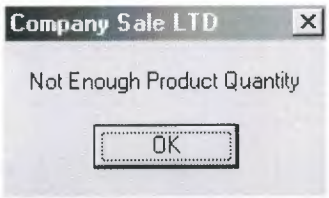


Fig 3.13 Warning About Quantity

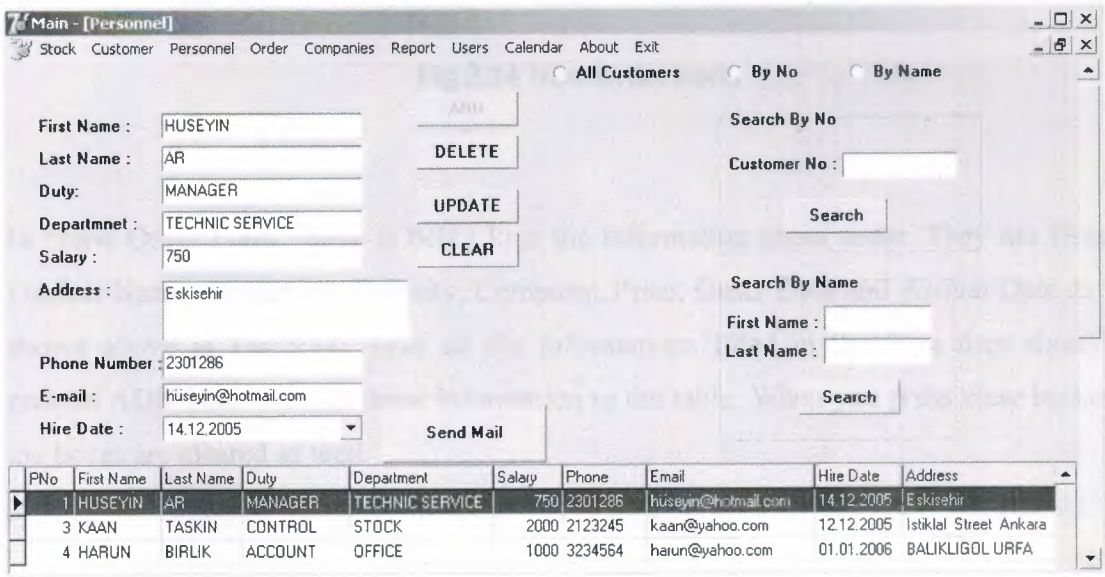


Fig 3.13 Personnel Form

Here Personnel details are held in this form. We can add personel, delete personel, update personel in this form. Moreover we can search personel by No and by First Name and Surname in the same form in Fig 3.7. And the personnel can send e-mail to each other by selecting a person in the table.

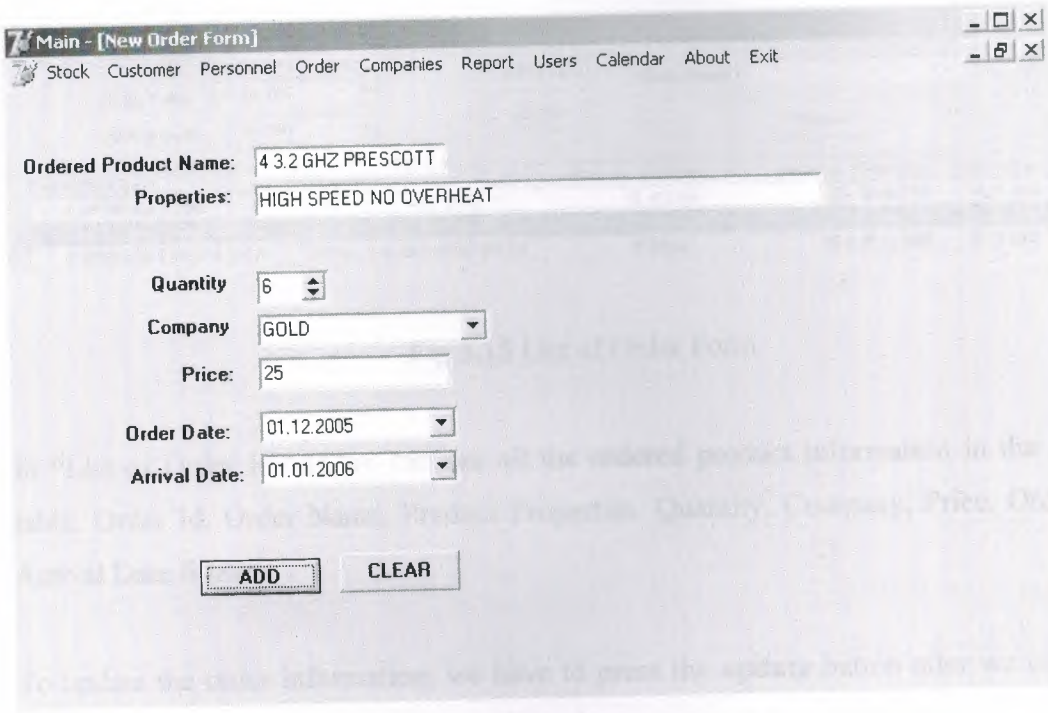


Fig 3.14 New Order Form

In "New Order Form" there is being kept the information about order. They are Ordered Product Name, Properties, Quantity, Company, Price, Order Date and Arrival Date as it is shown above in Fig 3.14. After all the informations filled in the form then should be pressed ADD button to add these information to the table. When you press clear button all the boxes are cleared as well.

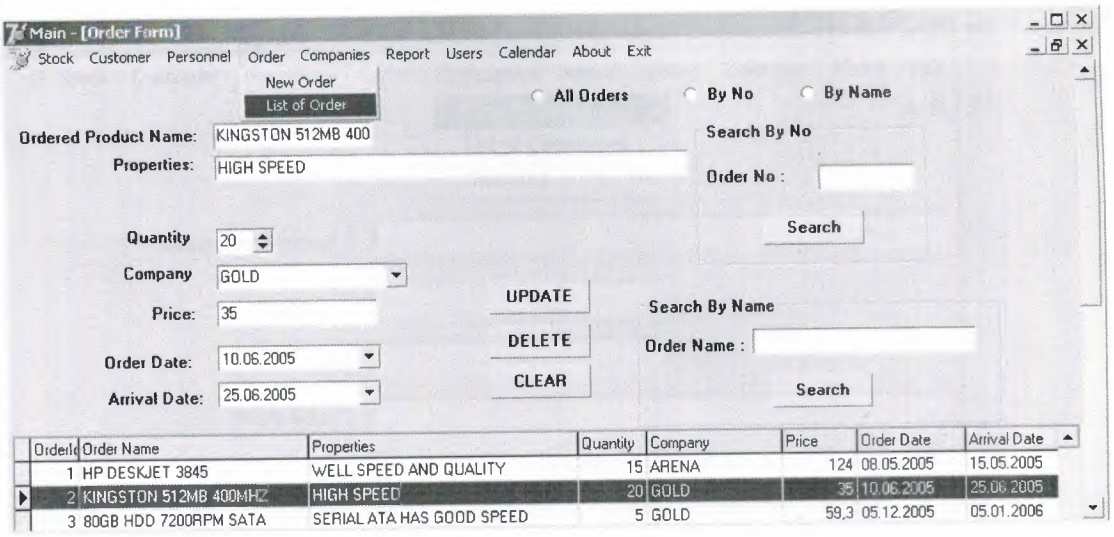


Fig 3.15 List of Order Form

In "List of Order Form" we can see all the ordered product information in the database table. Order Id, Order Name, Product Properties, Quantity, Company, Price, Order Date, Arrival Date fidelds.

To update the order information, we have to press the **update** button after we correct the information in the EditBox and ComboBox shown above in Fig. 3.15.

To delete the order, we have to press the **delete** button after we select the order product in the databae table.

To Search a product. We can search by Order No and search by Order Name after it is written Order No or Order Name.

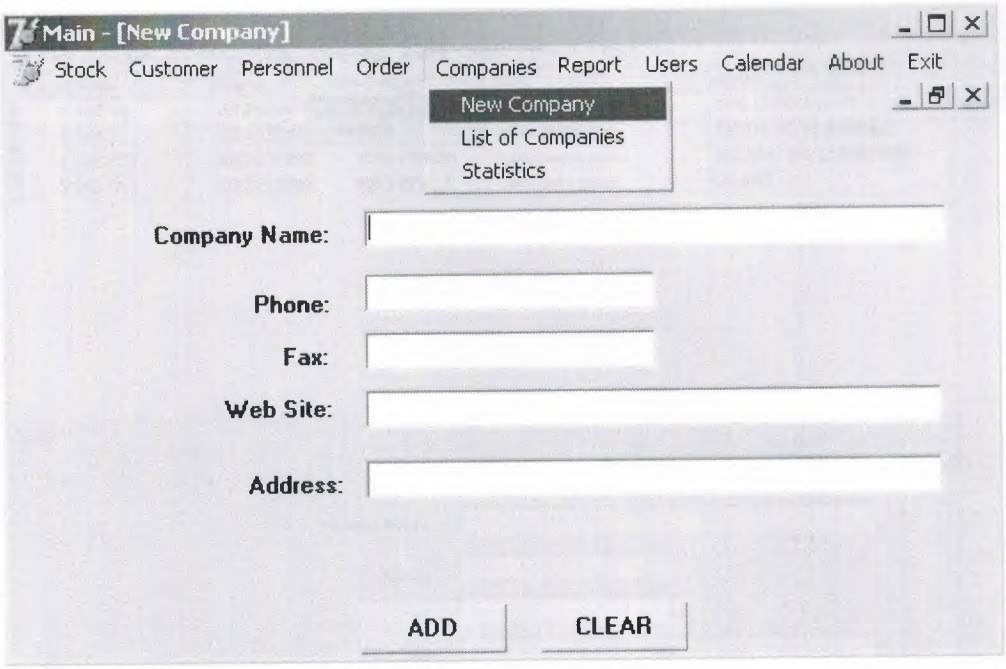


Fig 3.15 New Company Form

In "New Company Form " we can add new company to the database by pressing **add** button, after all the information are filled, from which company we are buying the product. And to clear the EditBox it is needed to press **clear** button.

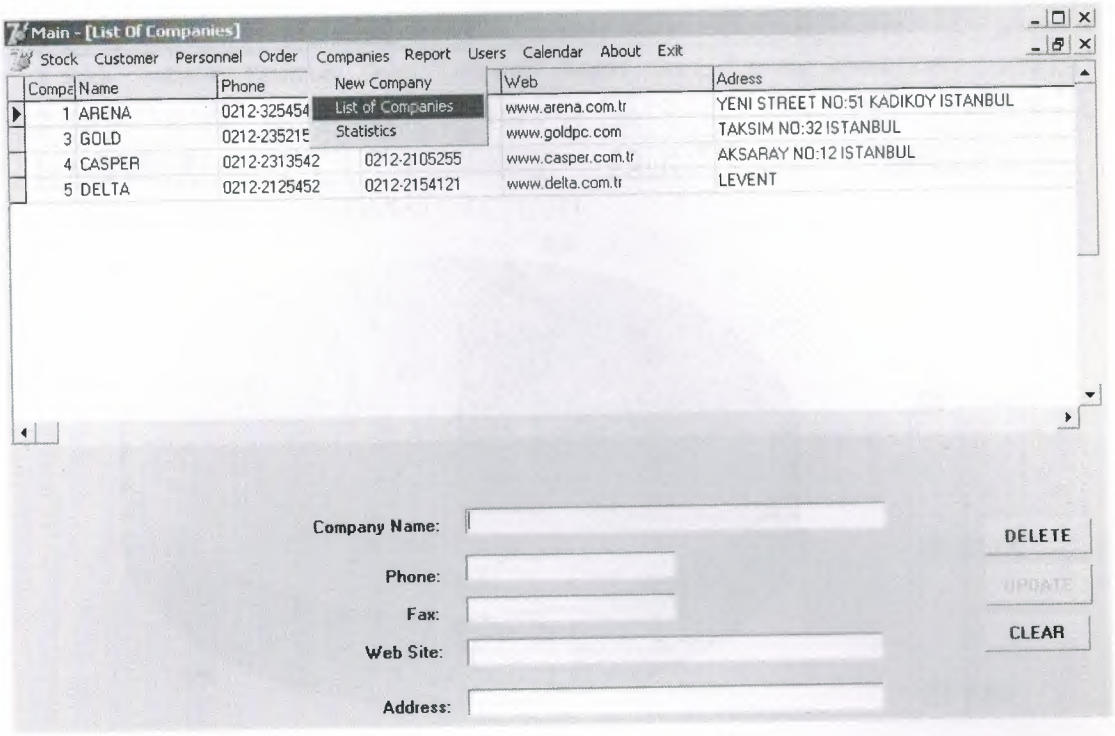


Fig 3.16 List of Companies Form

In "List of Companies Form " we can update and delete the companies which is needed. To update the company information, first we select the company on the table then correct the information in the EditBox and ComboBox shown above in Fig 3.16. above, then it is needed to press **update** button.

To delete the order, first it is needed to select a company in the database table then it is necessary to press **delete** button.

And to clear the EditBox it is needed to press **clear** button.

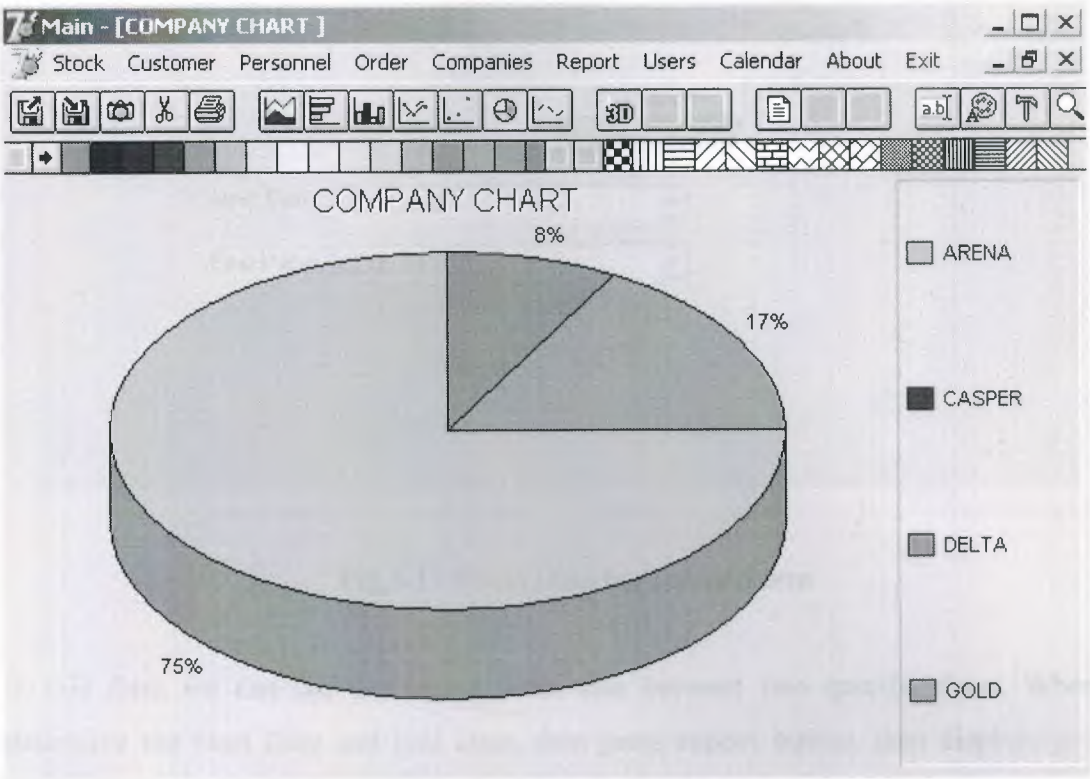


Fig 3.16 Chart

Company chart includes number of work done with the companies in a percentage value. The company names are colored and listed in the right hand side of chartfx control. Above there are toolbox, palette bar, pattern bar which give user to customize the chart by changing the color of each slices of the chart which is in pie type chart control. And also the chart type, 3d vision, show/hide list can be changed from the toolbar. Furthermore toolbar allows users to save, import, export or print the chart by clicking the proper buttons on the toolbar.

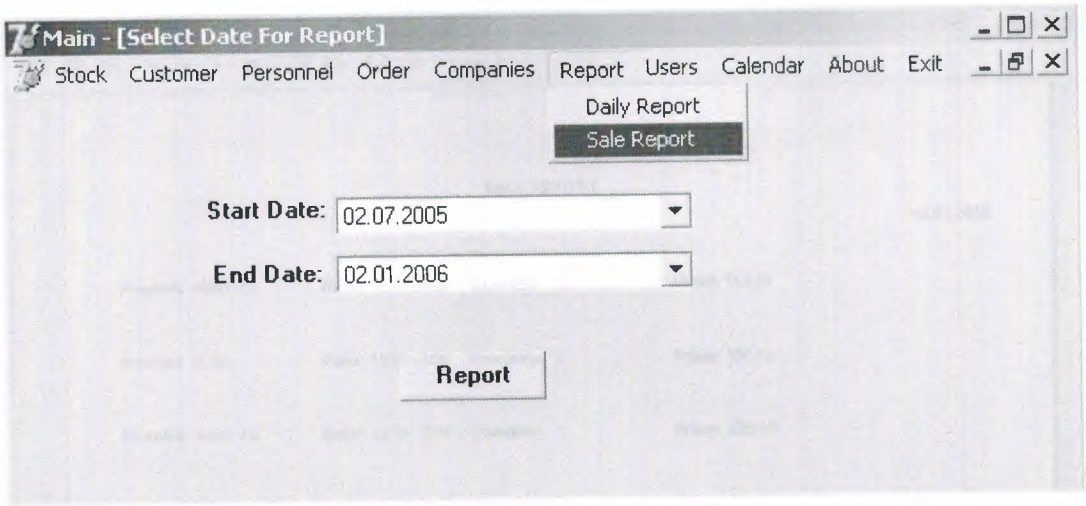


Fig 3.17 Select Date For Report Form

In this form we can see the report about sale between two specific dates. When we determine the Start Date and End Date, then press **report** button, then displays product name, date, price, quantity information as it is shown fig. below.

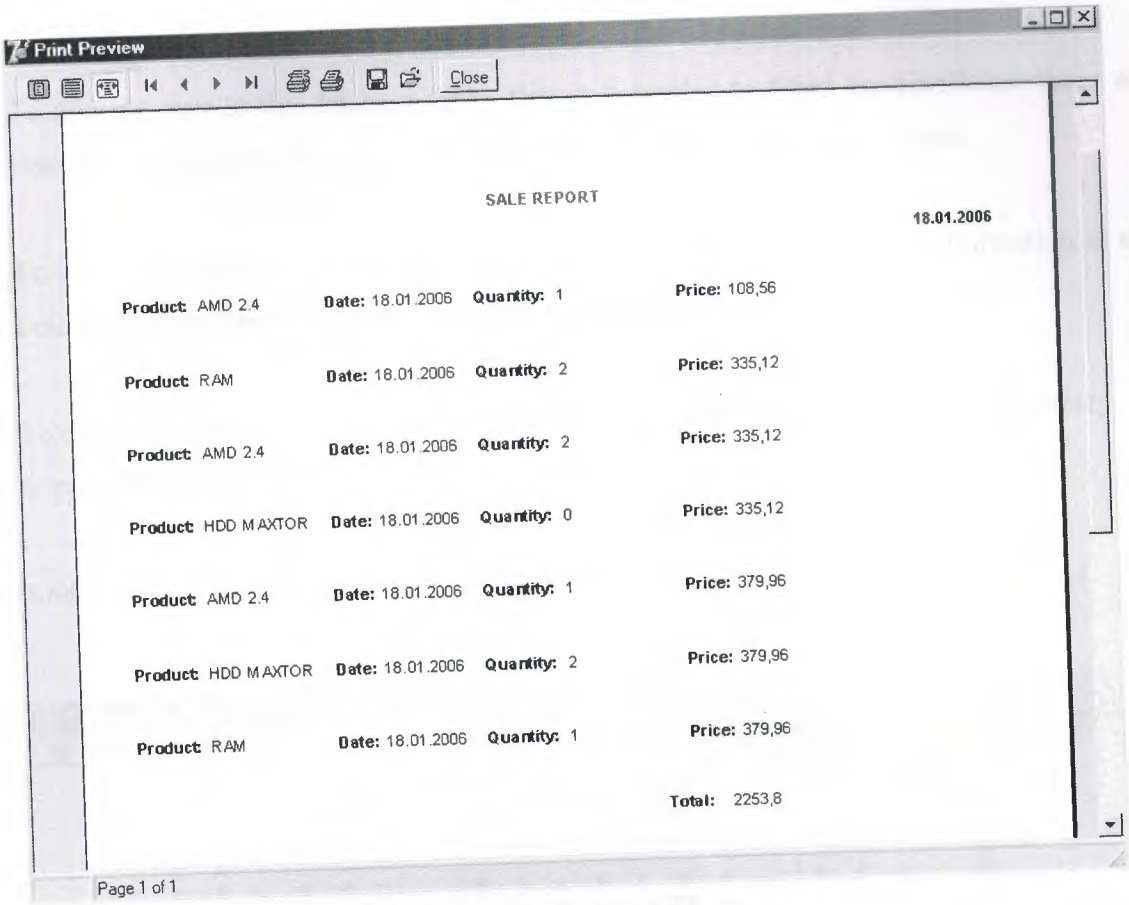


Fig 3.18 Report

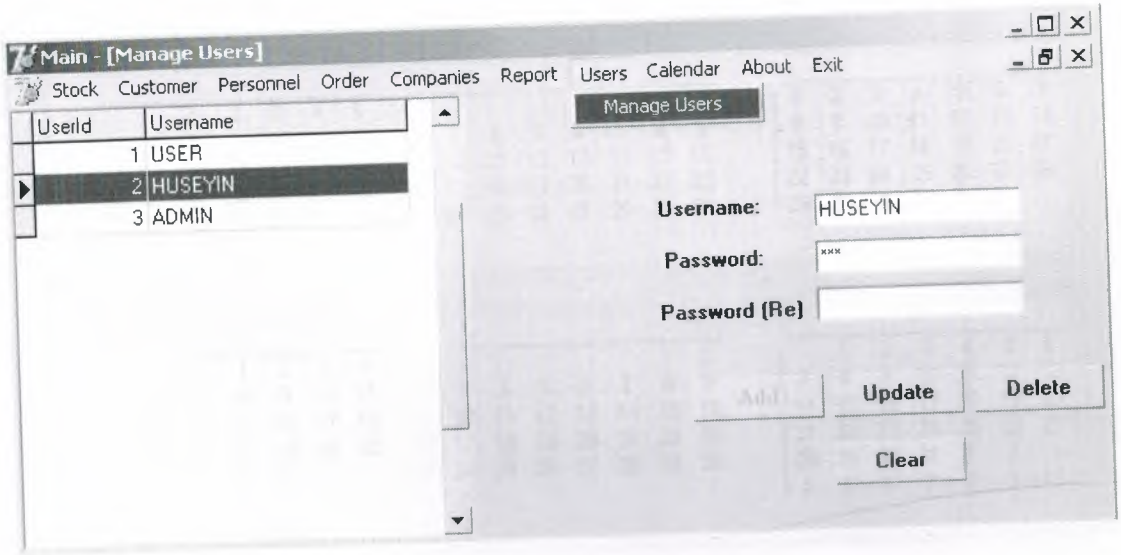


Fig 3.18 Manage User Form

In "Manage User Form" we can add user, update user and delete user.

To add the user to the database table, first it is needed to filled the EditBoxes those are Username, Password, Password(Re), then it is necessary to press **add** button.

To update the user, first we select the user in the table, then correct the information in the EditBox shown above in Fig. above, then it is needed to press **update** button.

To delete user, first it is needed to select a user in the database table, then it is necessary to press **DELETE** button.

And to clear the EditBoxes it is needed to press **clear** button.

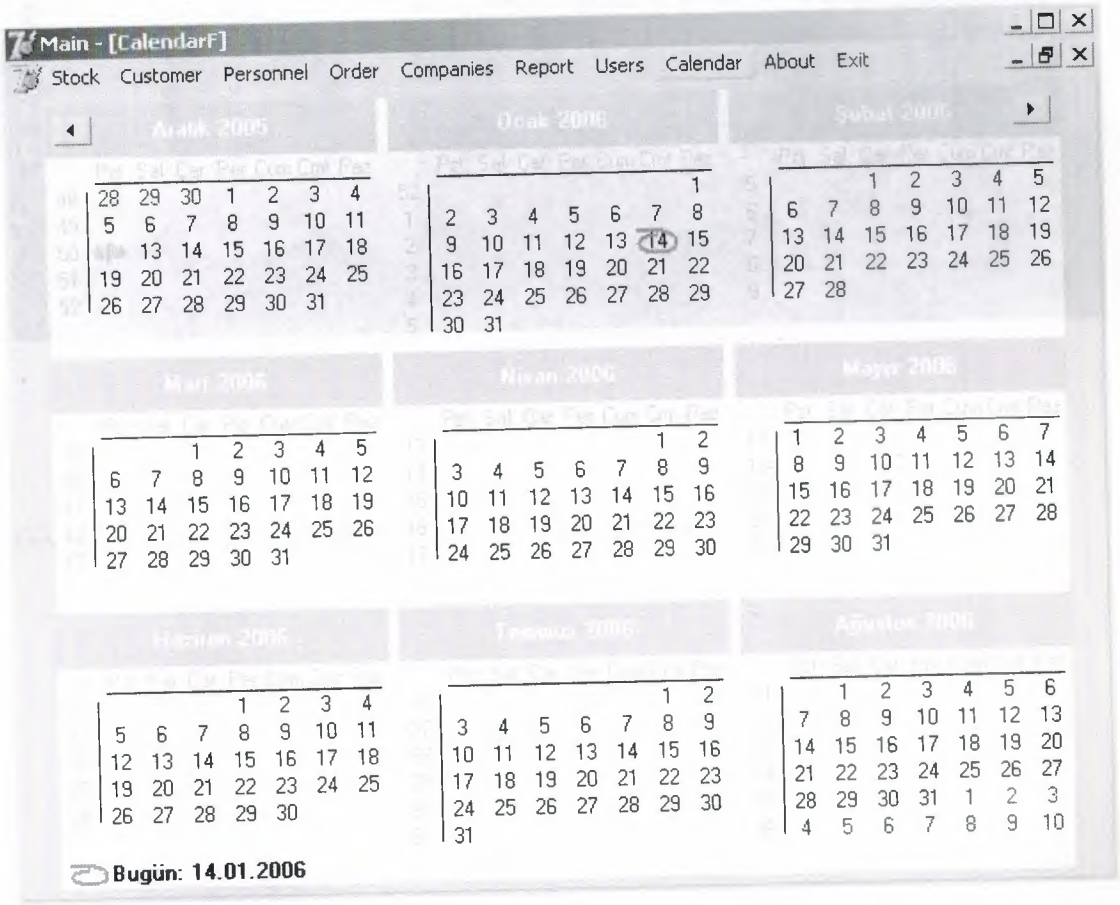


Fig 3.19 Calendar Form

In Calendar Form there is shown the Calendar as we see above Fig 3.19. We can see the date or specific date which we want to learn.

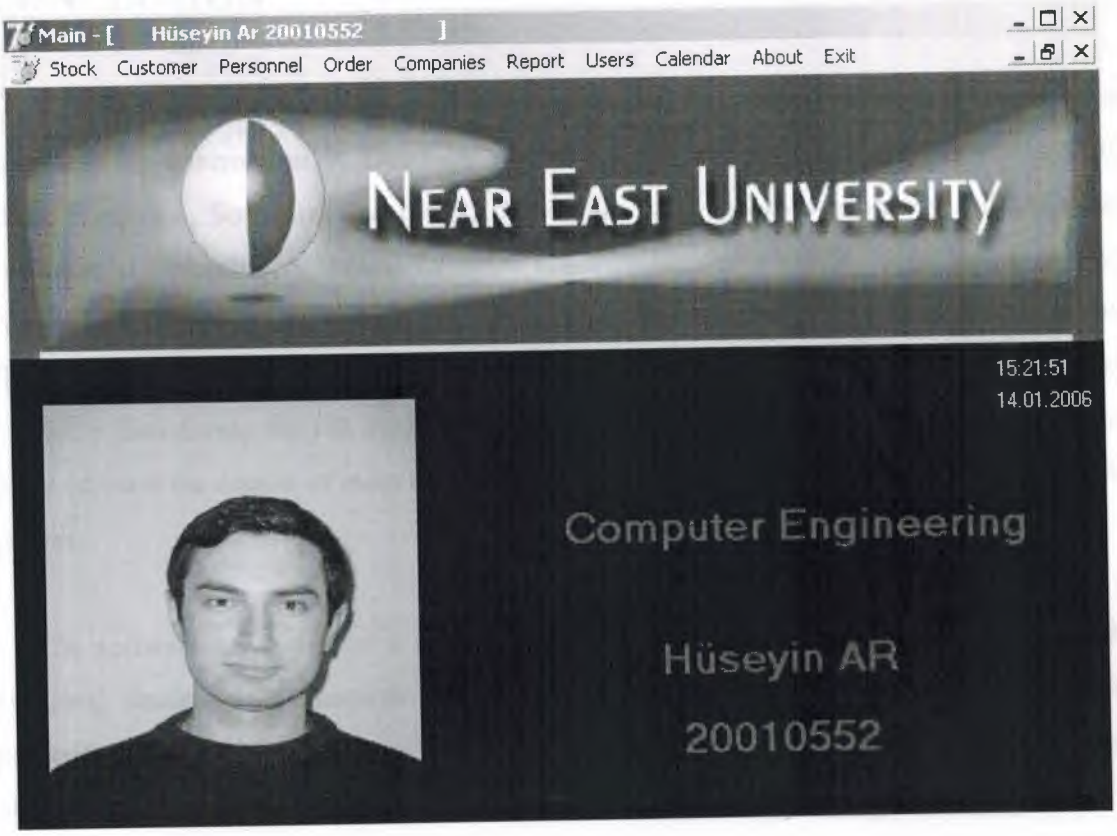


Fig 3.19 About Form

Gives the user a brief description about the programmer.

CONCLUSION

A DBMS is computerized record keeping system that stores, maintains and provides access to the information. A Database system consists of four major components that are Data, Hardware, Software, and Users. DBMS are used by any reasonably self contained commercial, scientific, technical or other organization for a single individual to a large company. Practically implementation of software for business though it is related to any field needs a devoted and complete life cycle. The most important idea is the attitude which has to be face during the life cycle of the Company or Organization. And according to this point of view the reason of most unsuccessful project is misunderstanding between the two parties.

The software created after a deep analysis, so that all-important requirements to the company dealing with computer sales and purchase can be accomplished. Company and product, name and ID have been included in the program to overcome the mistakes, which can occur. Reports are also generated with the help of the Queries for the update purpose.

REFERENCES

Reference to Book:

[1] Turkmen Kitabevi, Borland Delphi 7

Reference to Electronic-Book:

[1] Delphi 7 Delphi QuickStart .PDF

Reference to Electronic Source- Online source from Web:

[1] http://www. borland.com

APPENDIX A: PROGRAM CODES

FORM 2. PRODUCT ENTRY FORM

```
unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, Spin, ComCtrls, ExtCtrls, Grids,
  DBGrids, DB, DBTables;
type
  TForm2 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Add: TButton;
    Clean: TButton;
    Delete: TButton;
    Table1: TTable;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    Label9: TLabel;
    Label10: TLabel;
    ComboBox1: TComboBox;
```

```
ComboBox2: TComboBox;
code: TEdit;
Sname: TEdit;
price: TEdit;
Sprice: TEdit;
quantity: TSpinEdit;
guarantee: TSpinEdit;
date1: TDateTimePicker;
Update: TButton;
DBNavigator1: TDBNavigator;
Company: TComboBox;
compT: TTable;
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormActivate(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
procedure RadioButton1Click(Sender: TObject);
procedure AddClick(Sender: TObject);
procedure DeleteClick(Sender: TObject);
procedure CleanClick(Sender: TObject);
procedure quantityChange(Sender: TObject);
procedure guaranteeChange(Sender: TObject);
procedure ComboBox2Change(Sender: TObject);
procedure DBGrid1CellClick(Column: TColumn);
procedure UpdateClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure ComboBox1Change(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

```
var
 Form2: TForm2;
implementation
uses unit3;
{$R *.dfm}

procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
//Close Table1 on form close
 table1.Close;
 action:=caFree;
end;
procedure TForm2.FormActivate(Sender: TObject);
begin
//On form active make the window maximize
WindowState := wsMaximized;
end;

procedure TForm2.RadioButton2Click(Sender: TObject);
var    k:integer;
 i:integer;
begin
//Clear all fields on form
  clean.Click;
  Combobox1.Clear;
  Combobox2.Clear;
  Combobox1.Enabled := True;
  Combobox2.Enabled := True;
  //List All Products in Combobox1 and Combobox2
  table1.First;
  for i:=0 to Table1.RecordCount-1 do
```

```
begin
combobox1.Items.Add(table1.FieldValues['Stock_Name']);
combobox2.Items.Add(table1.FieldValues['Stock_Id']);
table1.Next;
end;
end;

procedure TForm2.RadioButton1Click(Sender: TObject);
begin
//combobox1 and combobox2 will be disabled
Combobox1.Enabled := False;
Combobox2.Enabled := False;
//Clear all fields on form
clean.Click;
end;

procedure TForm2.AddClick(Sender: TObject);
begin
// Add if New Product is selected
if RadioButton1.Checked=true then
 begin
//insert mode to add new record to table
  Table1.insert;
  Table1.FieldValues['Stock_Code']:=code.Text;
  Table1.FieldValues['Stock_Name']:=Sname.Text;
  Table1.FieldValues['Purchase_Price']:=strtofloat(price.Text);
  Table1.FieldValues['Selling_Price']:=strtofloat(Sprice.Text);
  Table1.FieldValues['Company']:=Company.Text;
  Table1.FieldValues['Quantity']:=strtoint(Quantity.Text);
  Table1.FieldValues['Guarantee']:=strtoint(Guarantee.Text);
  Table1.FieldValues['Date']:=DateToStr(date1.Date);
```

```
 Table1.Post;
 //Select Product in stock to list all products again
  radiobutton2.OnClick(sender);
  clean.Click;
  table1.close;
  table1.Open;
 end;
end;

procedure TForm2.DeleteClick(Sender: TObject);
begin
//Make user confirm deletion process
if Messagedlg('Please Confirm Deletion',mtwarning, [mbYes, mbNo,mbCancel],0)=6 then
  begin
  //if accepts delete record
  Table1.Delete;
  //list and refresh the product comboboxes
  radiobutton2.OnClick(sender);
  table1.Close;
  table1.Open;
  clean.Click;
  Showmessage('Deletion is successfull');
  end
  else
   Showmessage('Deletion is Cancelled');
 end;

 procedure TForm2.CleanClick(Sender: TObject);
 begin
 //Clear all fields on form
 code.Text:='';
```

```
Sname.Text:='';
price.Text:='';
Sprice.Text:='';
Company.Text:='';
quantity.Text:='0';
guarantee.Text:='0';
ADD.Enabled :=true;
Update.Enabled :=false;
end;

procedure TForm2.quantityChange(Sender: TObject);
begin
//if quantity is less then 0 make it 0
  if Quantity.Value<0 then
  Quantity.Value:=0;
end;

procedure TForm2.guaranteeChange(Sender: TObject);
begin
//if Guarantee is less then 0 make it 0
    if Guarantee.Value<0 then
    Guarantee.Value:=0;
end;

procedure TForm2.ComboBox2Change(Sender: TObject);
var     k,i:integer;
begin
//assign text of combobox2 to variable k
  k:=strtoint(Combobox2.Text);
  //set combobox1 index from the  variable k
  Combobox1.ItemIndex:=k-1;
```

```
Table1.open;
 //Find Record which has k id
table1.Locate('Stock_code',k,[loPartialKey]);
 //bring values to the fields on form
Sname.Text:=Table1.FieldValues['Stock_Name'];
Code.Text:=Table1.FieldValues['Stock_Code'];
Price.Text:=Table1.FieldValues['Purchase_Price'];
Sprice.Text:=Table1.FieldValues['Selling_Price'];
Company.Text:=Table1.FieldValues['Company'];
Guarantee.Text:=Table1.FieldValues['Guarantee'];
Quantity.Text:='0';
Date1.Date:=Table1.FieldValues['Date'];
end;

procedure TForm2.DBGrid1CellClick(Column: TColumn);
begin
if dbgrid1.Fields[dbgrid1.SelectedIndex].AsString<>'' then
begin
//bring values to the fields on form
Sname.Text := table1.FieldValues['Stock_Name'];
code.Text := table1.FieldValues['Stock_Code'];
Price.Text := table1.FieldValues['Purchase_Price'];
SPrice.Text := table1.FieldValues['Selling_Price'];
Company.Text := table1.FieldValues['Company'];
Guarantee.Text := table1.FieldValues['Guarantee'];
Date1.date := table1.FieldValues['Date'];
Quantity.Text := table1.FieldValues['Quantity'];
//disable add button and enable update button
Add.Enabled :=false;
Update.Enabled :=true;
end;
```

```
end;

procedure TForm2.UpdateClick(Sender: TObject);
begin
//Edit mode to update records
table1.Edit;
Table1.FieldValues['Stock_Code'] :=code.Text ;
table1.FieldValues['Stock_Name']:= Sname.Text;
 table1.FieldValues['Purchase_Price'] :=Price.Text;
 table1.FieldValues['Selling_Price'] :=SPrice.Text;
table1.FieldValues['Company']:=Company.Text ;
table1.FieldValues['Guarantee'] :=Guarantee.Text;
table1.FieldValues['Date']:=Date1.date ;
table1.FieldValues['Quantity']:=Quantity.Text ;
// send values to table
table1.Post;
//close table
table1.Close;
table1.Open;
end;

procedure TForm2.FormCreate(Sender: TObject);
var i : integer;
begin
table1.Open;
CompT.Open;
CompT.First;
//list all companies in the company combobox
for i:=0 to compT.RecordCount-1 do
begin
 company.Items.Add(compT.FieldValues['Name']);
```

```
 compT.Next;
end;
Compt.close;
end;

procedure TForm2.ComboBox1Change(Sender: TObject);
begin
//Find the selected product and bring values to the fields on form
 table1.Locate('Stock_name',combobox1.Text,[loPartialKey]);
Sname.Text:=Table1.FieldValues['Stock_Name'];
Code.Text:=Table1.FieldValues['Stock_Code'];
Price.Text:=Table1.FieldValues['Purchase_Price'];
Sprice.Text:=Table1.FieldValues['Selling_Price'];
Company.Text:=Table1.FieldValues['Company'];
Guarantee.Text:=Table1.FieldValues['Guarantee'];
Quantity.Text:='0';
Date1.Date:=Table1.FieldValues['Date'];
combobox2.ItemIndex := combobox2.Items.IndexOf(Table1.FieldValues['Stock_Id'] );
end;
end.
```

FORM 4. STOCK LIST FORM

```
unit Unit4;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Spin, ComCtrls, StdCtrls, Mask, DBCtrls, DB, DBTables, Grids,
  DBGrids, ExtCtrls;
```

```
type
 TForm4 = class(TForm)
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  Label6: TLabel;
  Label7: TLabel;
  Label8: TLabel;
  DBGrid1: TDBGrid;
  DataSource1: TDataSource;
  Table1: TTable;
  Panel1: TPanel;
  All: TRadioButton;
  Company: TRadioButton;
  Byname: TRadioButton;
  ByCode: TRadioButton;
  ByNameP: TPanel;
  PName: TEdit;
  SearchByNameB: TButton;
  Label9: TLabel;
  ByComp: TPanel;
  CompName: TEdit;
  Label10: TLabel;
  SearchByComp: TButton;
  ByCodeP: TPanel;
  Code: TEdit;
  Label11: TLabel;
  SearchByCode: TButton;
```

```
Scode: TEdit;
Sname: TEdit;
Pprice: TEdit;
Sprice: TEdit;
quantity: TEdit;
guarantee: TEdit;
date1: TEdit;
company1: TComboBox;
compT: TTable;
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormActivate(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure AllClick(Sender: TObject);
procedure ByCodeClick(Sender: TObject);
procedure CompanyClick(Sender: TObject);
procedure BynameClick(Sender: TObject);
procedure SearchByNameBClick(Sender: TObject);
procedure SearchByCompClick(Sender: TObject);
procedure SearchByCodeClick(Sender: TObject);
procedure DBGrid1CellClick(Column: TColumn);
procedure PNameKeyPress(Sender: TObject; var Key: Char);
procedure CompNameKeyPress(Sender: TObject; var Key: Char);
procedure CodeKeyPress(Sender: TObject; var Key: Char);
private
  { Private declarations }
public
  { Public declarations }
end;

var
 Form4: TForm4;
```

```
implementation
{$R *.dfm}

procedure Clear(sender:Tobject);
begin
//Clear All Fields on form4
with form4 do
begin
Scode.clear;
Sname.Clear;
Pprice.Clear;
Sprice.Clear;
company1.ItemIndex :=0;
quantity.Clear;
guarantee.Clear;
date1.Clear;
end;
end;

procedure TForm4.FormClose(Sender: TObject; var Action: TCloseAction);
begin
//close table on form4 when form closes
 table1.Close;
  action:=caFree;
end;

procedure TForm4.FormActivate(Sender: TObject);
begin
WindowState := wsMaximized;
end;
```

```
procedure TForm4.FormCreate(Sender: TObject);
var i :integer;
begin
// open table1
 table1.Open;
 CompT.Open;
 //list all companies into the company1 combobox
 CompT.First;
for i:=0 to compT.RecordCount-1 do
begin
 company1.Items.Add(compT.FieldValues['Name']);
 compt.Next;
end;
compT.Close;
end;

procedure TForm4.AllClick(Sender: TObject);
begin
// set visible false for all panels and list all products
BycodeP.Visible:=false;
Bycomp.Visible := false;
Bynamep.Visible:=false;
table1.Filtered:=false;
end;

procedure TForm4.ByCodeClick(Sender: TObject);
begin
//When Bycode is selected show Bycode panel and setfocus to the code textbox
if bycode.Checked =true then
begin
bycodeP.Visible:=true;
```

```
Bycomp.Visible:=false;
bynamep.Visible :=false;
code.setfocus;
clear(sender);
end
else
bycodeP.Visible:=false;
end;

procedure TForm4.CompanyClick(Sender: TObject);
begin
//When Company is selected show Company panel and setfocus to the compname textbox
if company.Checked =true then
begin
Bycomp.Visible:=true;
bynamep.Visible :=false;
bycodep.Visible :=false;
Compname.SetFocus ;
Clear(sender);
end
else
bycomp.Visible:=false;
end;

procedure TForm4.BynameClick(Sender: TObject);
begin
//When Byname is selected show Byname panel and setfocus to the Pname textbox
if Byname.Checked =true then
begin
bynameP.Visible:=true;
Bycomp.Visible:=false;
```

```
bycodep.Visible :=false;
Pname.SetFocus ;
clear(sender);
end
else
bynameP.Visible:=false;
end;

//UserDefined procedure to bring values of a selected record to the fields on form
procedure Getinfo(sender:Tobject);
begin
form4.scode.Text:= form4.table1.FieldValues['Stock_code'];
form4.sname.Text:= form4.table1.FieldValues['Stock_Name'];
form4.Pprice.Text:= form4.table1.FieldValues['Purchase_Price'];
form4.Sprice.Text:= form4.table1.FieldValues['Selling_Price'];
form4.company1.Text:= form4.table1.FieldValues['Company'];
form4.guarantee.Text:= form4.table1.FieldValues['Guarantee'];
form4.date1.Text:= form4.table1.FieldValues['Date'];
form4.quantity.Text:= form4.table1.FieldValues['Quantity'];
end;

procedure TForm4.SearchByNameBClick(Sender: TObject);
begin
// search in table by stock name
table1.filter:='Stock_Name='+ quotedstr(Pname.Text );
table1.Filtered:=true;
//if no record found
if table1.RecordCount =0 then
//show user message
showmessage('No Record Found')
else
```

```
//bring values by calling the userdefined procedure
getinfo(form4);
Pname.Clear;
Pname.SetFocus;
end;

procedure TForm4.SearchByCompClick(Sender: TObject);
begin
// search in table by company
table1.filter:='Company='+ quotedstr(CompName.Text );
table1.Filtered:=true;
if table1.RecordCount =0 then
 showmessage('No Record Found')
 else
 //bring values by calling the userdefined procedure
 getinfo(form4);
 compname.Clear;
 compname.SetFocus;
end;

procedure TForm4.SearchByCodeClick(Sender: TObject);
begin
// search in table by code
table1.filter:='Stock_Code='+ quotedstr(Code.Text );
table1.Filtered:=true;
if table1.RecordCount =0 then
 showmessage('No Record Found')
 else
  //bring values by calling the userdefined procedure
 getinfo(form4);
 code.Clear;
```

```
 code.SetFocus;
end;

//bring values of selected in dbgrid record onto form
procedure TForm4.DBGrid1CellClick(Column: TColumn);
begin
if dbgrid1.Fields[dbgrid1.SelectedIndex].AsString<>'' then
begin
getinfo(form4);
end;
end;
procedure TForm4.PNameKeyPress(Sender: TObject; var Key: Char);
begin
if key=#13 then
SearchByNameB.Click;
end;

procedure TForm4.CompNameKeyPress(Sender: TObject; var Key: Char);
begin
if key=#13 then
SearchByComp.Click;
end;

procedure TForm4.CodeKeyPress(Sender: TObject; var Key: Char);
begin
if key=#13 then
SearchByCode.Click;
end;
end.
```

FORM 8. WHOLESALE FORM

```
unit Unit8;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, DB, DBTables, ExtCtrls, DBCtrls, Grids, DBGrids,
  ComCtrls, Mask, Spin,StrUtils;

type
  TForm8 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    DataSource1: TDataSource;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    CheckBox4: TCheckBox;
    CheckBox5: TCheckBox;
    CheckBox6: TCheckBox;
    CheckBox7: TCheckBox;
    CheckBox8: TCheckBox;
    CheckBox9: TCheckBox;
    CheckBox10: TCheckBox;
```

```
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
SpinEdit1: TSpinEdit;
SpinEdit2: TSpinEdit;
SpinEdit3: TSpinEdit;
SpinEdit4: TSpinEdit;
SpinEdit5: TSpinEdit;
SpinEdit6: TSpinEdit;
SpinEdit7: TSpinEdit;
SpinEdit8: TSpinEdit;
SpinEdit9: TSpinEdit;
Label24: TLabel;
Label25: TLabel;
Label26: TLabel;
Label27: TLabel;
Label28: TLabel;
Label29: TLabel;
Label30: TLabel;
Label31: TLabel;
Label32: TLabel;
Label33: TLabel;
Label39: TLabel;
Label40: TLabel;
```

```
Label41: TLabel;
Label42: TLabel;
Label43: TLabel;
Label44: TLabel;
Label45: TLabel;
Label46: TLabel;
Label47: TLabel;
Label48: TLabel;
Label54: TLabel;
Label55: TLabel;
total: TEdit;
totalv: TEdit;
Button3: TButton;
Button4: TButton;
Button5: TButton;
ComboBox1: TComboBox;
ComboBox2: TComboBox;
ComboBox3: TComboBox;
ComboBox4: TComboBox;
ComboBox5: TComboBox;
ComboBox6: TComboBox;
ComboBox7: TComboBox;
ComboBox8: TComboBox;
ComboBox9: TComboBox;
ComboBox10: TComboBox;
SpinEdit10: TSpinEdit;
Edit3: TEdit;
Edit4: TEdit;
Edit5: TEdit;
Edit6: TEdit;
Edit7: TEdit;
```

```
Edit8: TEdit;
Edit9: TEdit;
Edit10: TEdit;
Edit11: TEdit;
Edit12: TEdit;
Customer: TTable;
DataSource2: TDataSource;
SpinEdit11: TSpinEdit;
SpinEdit12: TSpinEdit;
SpinEdit13: TSpinEdit;
SpinEdit14: TSpinEdit;
SpinEdit15: TSpinEdit;
SpinEdit16: TSpinEdit;
SpinEdit17: TSpinEdit;
SpinEdit18: TSpinEdit;
SpinEdit19: TSpinEdit;
SpinEdit20: TSpinEdit;
Edit1: TEdit;
Edit2: TEdit;
Edit13: TEdit;
Edit14: TEdit;
Edit15: TEdit;
Edit16: TEdit;
Edit17: TEdit;
CustomerCustomerId: TAutoIncField;
CustomerName: TStringField;
CustomerSurname: TStringField;
CustomerAddress: TStringField;
CustomerPhone: TStringField;
CustomerEmail: TStringField;
CustomerFax: TStringField;
```

```
sales: TTable;
salesSaleId: TAutoIncField;
salesCustomerID: TFloatField;
salesProductId: TFloatField;
salesDate: TDateField;
salesPrice: TFloatField;
salesQuantity: TFloatField;
Label8: TLabel;
Label19: TLabel;
Label20: TLabel;
Label21: TLabel;
Label22: TLabel;
Label23: TLabel;
Label34: TLabel;
Label35: TLabel;
Label36: TLabel;
Label37: TLabel;
stock: TTable;
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormCreate(Sender: TObject);
procedure CheckBox1Click(Sender: TObject);
procedure CheckBox2Click(Sender: TObject);
procedure CheckBox3Click(Sender: TObject);
procedure CheckBox4Click(Sender: TObject);
procedure CheckBox5Click(Sender: TObject);
procedure CheckBox6Click(Sender: TObject);
procedure CheckBox7Click(Sender: TObject);
procedure CheckBox8Click(Sender: TObject);
procedure CheckBox9Click(Sender: TObject);
procedure CheckBox10Click(Sender: TObject);
procedure ComboBox1Change(Sender: TObject);
```

```
procedure ComboBox2Change(Sender: TObject);
procedure ComboBox3Change(Sender: TObject);
procedure ComboBox4Change(Sender: TObject);
procedure ComboBox5Change(Sender: TObject);
procedure ComboBox6Change(Sender: TObject);
procedure ComboBox7Change(Sender: TObject);
procedure ComboBox8Change(Sender: TObject);
procedure ComboBox9Change(Sender: TObject);
procedure ComboBox10Change(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure SpinEdit1Change(Sender: TObject);
procedure SpinEdit2Change(Sender: TObject);
procedure SpinEdit3Change(Sender: TObject);
procedure SpinEdit4Change(Sender: TObject);
procedure SpinEdit5Change(Sender: TObject);
procedure SpinEdit6Change(Sender: TObject);
procedure SpinEdit7Change(Sender: TObject);
procedure SpinEdit8Change(Sender: TObject);
procedure SpinEdit9Change(Sender: TObject);
procedure SpinEdit10Change(Sender: TObject);
procedure SpinEdit11Change(Sender: TObject);
procedure SpinEdit12Change(Sender: TObject);
procedure SpinEdit13Change(Sender: TObject);
procedure SpinEdit14Change(Sender: TObject);
procedure SpinEdit15Change(Sender: TObject);
procedure SpinEdit16Change(Sender: TObject);
procedure SpinEdit17Change(Sender: TObject);
procedure SpinEdit18Change(Sender: TObject);
```

```
    procedure SpinEdit19Change(Sender: TObject);
    procedure SpinEdit20Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form8: TForm8;
  Vtotal:single;
 count:integer;

implementation

{$R *.dfm}

procedure TForm8.FormClose(Sender: TObject; var Action: TCloseAction);
begin
//close tables on form close
  Stock.Close;
  sales.Close;
    action:=caFree;
end;


procedure TForm8.FormCreate(Sender: TObject);
var i :integer;
begin
```

```
Vtotal:=0;
count:=0;
 Stock.open;
 //List Product names in the comboboxes
Stock.First;
for i:=0 to Stock.RecordCount-1 do
begin
 combobox1.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox2.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox3.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox4.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox5.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox7.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox6.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox8.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox9.Items.Add(Stock.FieldValues['Stock_Name']);
 combobox10.Items.Add(Stock.FieldValues['Stock_Name']);
 Stock.Next;
end;
sales.Open;
end;

procedure TForm8.CheckBox1Click(Sender: TObject);
begin
//If Checkbox1 is checked make proper combobox, spinedits and editboxes enable
  if checkbox1.Checked=true then
  begin
  combobox1.Enabled:=True;
  spinedit1.Enabled:=true;
  edit3.Enabled:=true;
  end
```

```
 else
 begin
  combobox1.Enabled:=false;
 spinedit1.Enabled:=false;
  edit3.Enabled:=false;
 end;

end;

procedure TForm8.CheckBox2Click(Sender: TObject);
begin
//If Checkbox2 is checked make proper combobox, spinedits and editboxes enable
 if checkbox2.Checked=true then
 begin
 combobox2.Enabled:=True;
 spinedit2.Enabled:=true;
   edit7.Enabled:=true;
 end
 else
 begin
 combobox2.Enabled:=False;
 spinedit2.Enabled:=False;
  edit7.Enabled:=false;
 end;
end;

procedure TForm8.CheckBox3Click(Sender: TObject);
begin
//If Checkbox3 is checked make proper combobox, spinedits and editboxes enable
 if checkbox3.Checked=true then
 begin
```

```
combobox3.Enabled:=True;
spinedit3.Enabled:=true;
 edit4.Enabled:=true;
end
else
begin
combobox3.Enabled:=false;
 spinedit3.Enabled:=false;
   edit4.Enabled:=false;
 end;
end;

procedure TForm8.CheckBox4Click(Sender: TObject);
begin
//If Checkbox4 is checked make proper combobox, spinedits and editboxes enable
  if checkbox4.Checked=true then
  begin
  combobox4.Enabled:=True;
  spinedit4.Enabled:=true;
    edit5.Enabled:=true;
  end
  else
  begin
  combobox4.Enabled:=false;
   spinedit4.Enabled:=false;
     edit5.Enabled:=false;
   end;
  end;

  procedure TForm8.CheckBox5Click(Sender: TObject);
  begin
```

```
//If Checkbox5 is checked make proper combobox, spinedits and editboxes enable
 if checkbox5.Checked=true then
 begin
 combobox5.Enabled:=True;
 spinedit5.Enabled:=true;
   edit6.Enabled:=true;
 end
 else
 begin
 combobox5.Enabled:=fALSe;
 spinedit5.Enabled:=false;
   edit6.Enabled:=false;
 end;
end;

procedure TForm8.CheckBox6Click(Sender: TObject);
begin
//If Checkbox6 is checked make proper combobox, spinedits and editboxes enable
  if checkbox6.Checked=true then
  begin
  combobox6.Enabled:=True;
  spinedit6.Enabled:=true;
    edit8.Enabled:=true;
  end
  else
  begin
 combobox6.Enabled:=false;
  spinedit6.Enabled:=false;
    edit8.Enabled:=false;
  end;
```

```
end;

procedure TForm8.CheckBox7Click(Sender: TObject);
begin
//If Checkbox7 is checked make proper combobox, spinedits and editboxes enable
 if checkbox7.Checked=true then
 begin
 combobox7.Enabled:=True;
 spinedit7.Enabled:=true;
   edit12.Enabled:=true;

 end
 else
 begin
combobox7.Enabled:=False;
 spinedit7.Enabled:=false;
   edit12.Enabled:=false;
 end;
end;

procedure TForm8.CheckBox8Click(Sender: TObject);
begin
//If Checkbox8 is checked make proper combobox, spinedits and editboxes enable
 if checkbox8.Checked=true then
 begin
 combobox8.Enabled:=True;
 spinedit8.Enabled:=true;
   edit9.Enabled:=true;
 end
 else
 begin
```

```
combobox8.Enabled:=False;
 spinedit8.Enabled:=false;
   edit9.Enabled:=false;
 end;
end;

procedure TForm8.CheckBox9Click(Sender: TObject);
begin
//If Checkbox9 is checked make proper combobox, spinedits and editboxes enable
  if checkbox9.Checked=true then
  begin
  combobox9.Enabled:=True;
  spinedit9.Enabled:=true;
    edit10.Enabled:=true;
  end
  else
  begin
  combobox9.Enabled:=False;
   spinedit9.Enabled:=False;
     edit10.Enabled:=false;
   end;

  end;

  procedure TForm8.CheckBox10Click(Sender: TObject);
  begin
  //If Checkbox10 is checked make proper combobox, spinedits and editboxes enable
    if checkbox10.Checked=true then
    begin
    combobox10.Enabled:=True;
    spinedit10.Enabled:=true;
```

```
  edit11.Enabled:=true;
end
else
begin
combobox10.Enabled:=False;
spinedit10.Enabled:=false;
  edit11.Enabled:=false;
end;
end;

procedure TForm8.ComboBox1Change(Sender: TObject);
begin
//When a product is selected from Combobox1 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name=' +quotedstr(combobox1.Text);
label3.Caption := combobox1.Text;
stock.Filtered:=true;
label8.Caption := stock.FieldValues['Stock_Id'];
Edit3.Text := stock.FieldValues['Selling_Price'];
Spinedit11.Value :=stock.FieldValues['Guarantee'];

end;

procedure TForm8.ComboBox2Change(Sender: TObject);
begin
//When a product is selected from Combobox2 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox2.Text);
stock.Filtered:=true;
label19.Caption := stock.FieldValues['Stock_Id'];
Edit7.Text := stock.FieldValues['Selling_Price'];
```

```
Spinedit12.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox3Change(Sender: TObject);
begin
//When a product is selected from Combobox3 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox3.Text);
stock.Filtered:=true;
label20.Caption := stock.FieldValues['Stock_Id'];
Edit4.Text := stock.FieldValues['Selling_Price'];
Spinedit13.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox4Change(Sender: TObject);
begin
//When a product is selected from Combobox4 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox4.Text);
stock.Filtered:=true;
label21.Caption := stock.FieldValues['Stock_Id'];
Edit5.Text := stock.FieldValues['Selling_Price'];
Spinedit14.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox5Change(Sender: TObject);
begin
//When a product is selected from Combobox5 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox5.Text);
stock.Filtered:=true;
```

```
label22.Caption := stock.FieldValues['Stock_Id'];
Edit6.Text :=stock.FieldValues['Selling_Price'];
Spinedit15.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox6Change(Sender: TObject);
begin
//When a product is selected from Combobox6 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox6.Text);
stock.Filtered:=true;
label23.Caption := stock.FieldValues['Stock_Id'];
Edit8.Text := stock.FieldValues['Selling_Price'];
Spinedit16.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox7Change(Sender: TObject);
begin
//When a product is selected from Combobox7 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox7.Text);
stock.Filtered:=true;
label34.Caption := stock.FieldValues['Stock_Id'];
Edit12.Text := stock.FieldValues['Selling_Price'];
Spinedit17.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox8Change(Sender: TObject);
begin
//When a product is selected from Combobox8 the price and guarantee is brought to the
proper fields
```

```
stock.Filter:='Stock_Name='+ quotedstr(combobox8.Text);
stock.Filtered:=true;
label35.Caption := stock.FieldValues['Stock_Id'];
Edit9.Text := stock.FieldValues['Selling_Price'];
Spinedit18.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox9Change(Sender: TObject);
begin
//When a product is selected from Combobox9 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox9.Text);
stock.Filtered:=true;
label36.Caption := stock.FieldValues['Stock_Id'];
Edit10.Text := stock.FieldValues['Selling_Price'];
Spinedit19.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.ComboBox10Change(Sender: TObject);
begin
//When a product is selected from Combobox10 the price and guarantee is brought to the
proper fields
stock.Filter:='Stock_Name='+ quotedstr(combobox10.Text);
stock.Filtered:=true;
label37.Caption := stock.FieldValues['Stock_Id'];
Edit11.Text := stock.FieldValues['Selling_Price'];
Spinedit20.Value :=stock.FieldValues['Guarantee'];
end;

procedure TForm8.Button4Click(Sender: TObject);
//Calculate the Total price of the selected products and quantities
```

```
begin
Vtotal:=0;
if edit3.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit1.Text)*strtofloat(edit3.text);
if edit7.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit2.Text)*strtofloat(edit7.text);
if edit4.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit3.Text)*strtofloat(edit4.text);
if edit5.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit4.Text)*strtofloat(edit5.text);
if edit6.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit5.Text)*strtofloat(edit6.text);
if edit8.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit6.Text)*strtofloat(edit8.text);
if edit12.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit7.Text)*strtofloat(edit12.text);
if edit9.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit8.Text)*strtofloat(edit9.text);
if edit10.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit9.Text)*strtofloat(edit10.text);
if edit11.Text <>'' then
  Vtotal :=Vtotal+strtoint(spinedit10.Text)*strtofloat(edit11.text);
//calculate the total and total+VAT
  Total.Text := floattostr(Vtotal);
  Totalv.Text := floattostr(Vtotal*1.18);
end;

procedure TForm8.Edit1Change(Sender: TObject);
begin
// Find and List Customer Informations on the form
Customer.Open;
```

```
customer.Filter:= 'CustomerId=' +edit1.Text ;
customer.Filtered :=true;
edit2.Text := customer.FieldValues['Name'];
edit13.Text := customer.FieldValues['Surname'];
edit14.Text := customer.FieldValues['Address'];
edit15.Text := customer.FieldValues['Phone'];
edit16.Text := customer.fieldvalues['Fax'];
edit17.Text := customer.FieldValues['Email'];
end;

procedure TForm8.Button3Click(Sender: TObject);
begin
//Clear the fields on form
total.Clear;
totalv.Clear;
vtotal:=0;
combobox1.Text :='';
combobox2.Text :='';
combobox3.Text :='';
combobox4.Text :='';
combobox5.Text :='';
combobox6.Text :='';
combobox7.Text :='';
combobox8.Text :='';
combobox9.Text :='';
combobox10.Text :='';
spinedit1.Value :=0;
spinedit2.Value :=0;
spinedit3.Value :=0;
spinedit4.Value :=0;
spinedit5.Value :=0;
```

```
spinedit6.Value :=0;
spinedit7.Value :=0;
spinedit8.Value :=0;
spinedit9.Value :=0;
spinedit10.Value :=0;
edit3.Clear;
edit7.Clear;
edit4.clear;
edit5.Clear;
edit6.Clear;
edit8.clear;
edit12.Clear;
edit9.Clear;
edit10.clear;
edit11.clear;
spinedit11.Value :=0;
spinedit12.Value :=0;
spinedit13.Value :=0;
spinedit14.Value :=0;
spinedit15.Value :=0;
spinedit16.Value :=0;
spinedit17.Value :=0;
spinedit18.Value :=0;
spinedit19.Value :=0;
spinedit20.Value :=0;
checkbox1.Checked :=false;
checkbox2.Checked :=false;
checkbox3.Checked :=false;
checkbox4.Checked :=false;
checkbox5.Checked :=false;
checkbox6.Checked :=false;
```

```
checkbox7.Checked :=false;
checkbox8.Checked :=false;
checkbox9.Checked :=false;
checkbox10.Checked :=false;

end;

procedure TForm8.Button5Click(Sender: TObject);
var i:integer;
 check :boolean;
 s:string;
begin
//Make user Confirm Sale
if Messagedlg('Please Confirm Sale',mtwarning, [mbYes, mbNo,mbCancel],0)=6 then
  begin

 check:=true;
 s:='';
 //Sell the selected products and decrease the quantity amount from the stock
 if checkbox1.Checked =true then
 begin

 stock.Filter:='Stock_Id='+quotedstr(label8.Caption);
 stock.Filtered:=true;
 if stock.FieldValues['Quantity']> spinedit1.Value then
 begin
 sales.Insert;
    sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
    sales.FieldValues['ProductId']:=strtoint(label8.Caption) ;
    sales.FieldValues['Date']:=now;
    sales.FieldValues['quantity']:=spinedit1.Value ;
```

```
  sales.FieldValues['Price']:=strtofloat(totalv.Text);
sales.Post;
stock.Edit;
stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit1.Value ;
stock.Post;
end
else
 begin
 // s variable is used to store the products that are not enough at the stock
 s:=s+stock.FieldValues['Stock_name']+', ';
  check:=false;
 end;
end;
if checkbox2.Checked =true then
begin

stock.Filter:='Stock_Id='+quotedstr(label19.Caption);
stock.Filtered:=true;
if stock.FieldValues['Quantity']> spinedit2.Value then
begin
sales.Insert;
   sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
   sales.FieldValues['ProductId']:=strtoint(label19.Caption);
   sales.FieldValues['Date']:=now;
   sales.FieldValues['quantity']:=spinedit2.Value ;
   sales.FieldValues['Price']:=strtofloat(totalv.Text);
 sales.Post;
 stock.Edit;
 stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit2.Value ;
 stock.Post;
 end
```

```
else
 begin
s:=s+stock.FieldValues['Stock_name']+', ';
 check:=false;
 end;
end;
if checkbox3.Checked =true then
begin

stock.Filter:='Stock_Id='+quotedstr(label20.Caption);
stock.Filtered:=true;
if stock.FieldValues['Quantity']>  spinedit3.Value then
begin
sales.Insert;
   sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
   sales.FieldValues['ProductId']:=strtoint(label20.Caption);
   sales.FieldValues['Date']:=now;
   sales.FieldValues['quantity']:=spinedit3.Value ;
   sales.FieldValues['Price']:=strtofloat(totalv.Text);
  sales.Post;
  stock.Edit;
  stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit3.Value ;
  stock.Post;
  end
  else
   begin
   s:=s+stock.FieldValues['Stock_name']+', ';
    check:=false;
    end;
   end;
   if checkbox4.Checked =true then
```

```
begin

stock.Filter:='Stock_Id='+quotedstr(label21.Caption);
stock.Filtered:=true;
if stock.FieldValues['Quantity']>  spinedit4.Value then
begin
sales.Insert;
   sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
   sales.FieldValues['ProductId']:=strtoint(label21.Caption);
   sales.FieldValues['Date']:=now;
   sales.FieldValues['quantity']:=spinedit4.Value ;
   sales.FieldValues['Price']:=strtofloat(totalv.Text);
sales.Post;
stock.Edit;
stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit4.Value ;
stock.Post;
end
else
  begin
 s:=s+stock.FieldValues['Stock_name']+',';
  check:=false;
 end;
end;
if checkbox5.Checked =true then
begin

stock.Filter:='Stock_Id='+quotedstr(label22.Caption);
stock.Filtered:=true;
if stock.FieldValues['Quantity']>  spinedit5.Value then
begin
sales.Insert;
```

```
  sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
  sales.FieldValues['ProductId']:=strtoint(label22.Caption);
  sales.FieldValues['Date']:=now;
  sales.FieldValues['quantity']:=spinedit5.Value ;
  sales.FieldValues['Price']:=strtofloat(totalv.Text);
sales.Post;
stock.Edit;
stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit5.Value ;
stock.Post;
end
else
 begin
 s:=s+stock.FieldValues['Stock_name']+',';
  check:=false;
 end;
end;
if checkbox6.Checked =true then
begin

stock.Filter:='Stock_Id='+quotedstr(label23.Caption);
stock.Filtered:=true;
if stock.FieldValues['Quantity']> spinedit6.Value then
begin
sales.Insert;
   sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
   sales.FieldValues['ProductId']:=strtoint(label23.Caption);
   sales.FieldValues['Date']:=now;
   sales.FieldValues['quantity']:=spinedit6.Value ;
   sales.FieldValues['Price']:=strtofloat(totalv.Text);
 sales.Post;
 stock.Edit;
```

```
stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit6.Value ;
stock.Post;
end
else
 begin
 s:=s+stock.FieldValues['Stock_name']+',';
  check:=false;
 end;
end;
if checkbox7.Checked =true then
begin

 stock.Filter:='Stock_Id='+quotedstr(label34.Caption);
 stock.Filtered:=true;
 if stock.FieldValues['Quantity']>  spinedit7.Value then
 begin
 sales.Insert;
    sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
    sales.FieldValues['ProductId']:=strtoint(label34.Caption);
    sales.FieldValues['Date']:=now;
    sales.FieldValues['quantity']:=spinedit7.Value ;
    sales.FieldValues['Price']:=strtofloat(totalv.Text);
  sales.Post;
  stock.Edit;
  stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit7.Value ;
  stock.Post;
  end
  else
    begin
   s:=s+stock.FieldValues['Stock_name']+', ';
    check:=false;
```

```
 end;
end;
if checkbox8.Checked =true then
begin

stock.Filter:='Stock_Id='+quotedstr(label35.Caption);
stock.Filtered:=true;
if stock.FieldValues['Quantity']>  spinedit8.Value then
begin
sales.Insert;
   sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
   sales.FieldValues['ProductId']:=strtoint(label35.Caption);
   sales.FieldValues['Date']:=now;
   sales.FieldValues['quantity']:=spinedit8.Value ;
   sales.FieldValues['Price']:=strtofloat(totalv.Text);
sales.Post;
stock.Edit;
stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit8.Value ;
stock.Post;
end
else
  begin
 s:=s+stock.FieldValues['Stock_name']+', ';
  check:=false;
 end;
end;
if checkbox9.Checked =true then
begin

stock.Filter:='Stock_Id='+quotedstr(label36.Caption);
stock.Filtered:=true;
```

```
if stock.FieldValues['Quantity']>  spinedit9.Value then
begin
sales.Insert;
   sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
   sales.FieldValues['ProductId']:=strtoint(label36.Caption);
   sales.FieldValues['Date']:=now;
   sales.FieldValues['quantity']:=spinedit9.Value ;
   sales.FieldValues['Price']:=strtofloat(totalv.Text);
sales.Post;
stock.Edit;
stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit9.Value ;
stock.Post;
end
else
 begin
 s:=s+stock.FieldValues['Stock_name']+', ';
  check:=false;
 end;
end;
if checkbox10.Checked =true then
begin

stock.Filter:='Stock_Id='+quotedstr(label37.Caption);
stock.Filtered:=true;
if stock.FieldValues['Quantity']>  spinedit10.Value then
begin
sales.Insert;
   sales.FieldValues['CustomerId']:=strtoint(edit1.Text);
   sales.FieldValues['ProductId']:=strtoint(label37.Caption);
   sales.FieldValues['Date']:=now;
   sales.FieldValues['quantity']:=spinedit10.Value ;
```

```
  sales.FieldValues['Price']:=strtofloat(totalv.Text);
sales.Post;
stock.Edit;
stock.FieldValues['Quantity']:= stock.FieldValues['Quantity']-spinedit10.Value ;
stock.Post;
end
else
 begin
 s:=s+stock.FieldValues['Stock_name']+', ';
  check:=false;
 end;
end;
sales.Close;
 if check=true then
 showmessage('Sale is successfull')
 else
 begin
 // Prepare the message and Show to the user
s:=MidStr(s,1,length(s)-2);
s:=s+' ';
 showmessage(s+'Products is not enough in stock');
 end;
end;
end;

//Check the spinedits value
//If negative make the spinedits'
//value = 0
procedure TForm8.SpinEdit1Change(Sender: TObject);
begin
if spinedit1.Text <>'' then
```

```
begin
if spinedit1.Value <0 then
spinedit1.Value :=0;
end;
end;

procedure TForm8.SpinEdit2Change(Sender: TObject);
begin
if spinedit2.Text <>'' then
begin
if spinedit2.Value <0 then
spinedit2.Value :=0;
end;
end;

procedure TForm8.SpinEdit3Change(Sender: TObject);
begin
if spinedit3.Text <>'' then
begin
if spinedit3.Value <0 then
spinedit3.Value :=0;
end;
end;

procedure TForm8.SpinEdit4Change(Sender: TObject);
begin
if spinedit4.Text <>'' then
begin
if spinedit4.Value <0 then
spinedit4.Value :=0;
end;
```

```
end;

procedure TForm8.SpinEdit5Change(Sender: TObject);
begin
if spinedit5.Text <>'' then
begin
if spinedit5.Value <0 then
spinedit5.Value :=0;
end;
end;

procedure TForm8.SpinEdit6Change(Sender: TObject);
begin
if spinedit6.Text <>'' then
begin
if spinedit6.Value <0 then
spinedit6.Value :=0;
end;
end;

procedure TForm8.SpinEdit7Change(Sender: TObject);
begin
if spinedit7.Text <>'' then
begin
if spinedit7.Value <0 then
spinedit7.Value :=0;
end;
end;

procedure TForm8.SpinEdit8Change(Sender: TObject);
begin
```

```
if spinedit8.Text <>'' then
begin
if spinedit8.Value <0 then
spinedit8.Value :=0;
end;
end;

procedure TForm8.SpinEdit9Change(Sender: TObject);
begin
if spinedit9.Text <>'' then
begin
if spinedit9.Value <0 then
spinedit9.Value :=0;
end;
end;

procedure TForm8.SpinEdit10Change(Sender: TObject);
begin
if spinedit10.Text <>'' then
begin
if spinedit10.Value <0 then
spinedit10.Value :=0;
end;
end;

procedure TForm8.SpinEdit11Change(Sender: TObject);
begin
if spinedit11.Value <0 then
spinedit11.Value :=0;
end;
```

```
procedure TForm8.SpinEdit12Change(Sender: TObject);
begin
if spinedit12.Value <0 then
spinedit12.Value :=0;
end;

procedure TForm8.SpinEdit13Change(Sender: TObject);
begin
if spinedit13.Value <0 then
spinedit13.Value :=0;
end;

procedure TForm8.SpinEdit14Change(Sender: TObject);
begin
if spinedit14.Value <0 then
spinedit14.Value :=0;
end;

procedure TForm8.SpinEdit15Change(Sender: TObject);
begin
if spinedit15.Value <0 then
spinedit15.Value :=0;
end;

procedure TForm8.SpinEdit16Change(Sender: TObject);
begin
if spinedit16.Value <0 then
spinedit16.Value :=0;
end;

procedure TForm8.SpinEdit17Change(Sender: TObject);
```

```
begin
if spinedit17.Value <0 then
spinedit17.Value :=0;
end;

procedure TForm8.SpinEdit18Change(Sender: TObject);
begin
if spinedit18.Value <0 then
spinedit18.Value :=0;
end;

procedure TForm8.SpinEdit19Change(Sender: TObject);
begin
if spinedit19.Value <0 then
spinedit19.Value :=0;
end;

procedure TForm8.SpinEdit20Change(Sender: TObject);
begin
if spinedit20.Value <0 then
spinedit20.Value :=0;
end;
end.
```

APPENDIX B: DATABASE TABLES

Database Desktop - [Table : C:\Company\login.db]

File Edit View Table Record Tools Window Help

login	UserId	UName	UPass
1	1	USER	USER
2	2	HUSEYIN	123
3	3	ADMIN	123

Record 1 of 3

Fig. 4.1 Login Table

Database Desktop

File Edit View Table Record Tools Window Help

Table : C:\Company\Stock_List.DB

Stock_List	Stock_Id	Stock_Code	Stock_Name	Purchase_Price	Selling_Price	Company	Guaran	Date	Quantity
1	1	CPU	AMD 2.4	76,00	92,00	ARENA	2	02.06.2005	8
2	2	HDD	HDD MAXTOR	75,00	90,00	ARENA	2	03.02.2004	18
3	3	DVD-RW	CD/DVD-->LG 16X DOUBLE	76,00	92,00	ARENA	2	02.06.2005	16
4	4	RAM DDR2	RAM	35,00	50,00	DOLD	5	03.02.2005	20
5	6	DVD-RW	CD/DVD-->LG 16X DOUBLE	76,00	92,00	ARENA	2	02.06.2005	16
6	7	DVD-RW	CD/DVD-->LG 16X DOUBLE	76,00	92,00	ARENA	2	02.06.2005	16
7	8	CPU1	CPU-->INTEL P4 3.2 GHZ P	226,00	266,00	ARENA	3	11.05.2005	25
8	9	HDD1	HARDDISK-->120GB HDD 7	83,00	100,00	DUNYA	2	03.05.2005	28
9	10	SCREEN	SCREEN CARD-->MSI ATI I	45,45	60,00	DELTA	2	03.05.2005	24
10	11	MON1	MONITOR-->PHILIPS 17" 1C	132,30	165,00	HISBIM	3	06.06.2005	21

Record 1 of 15

Fig. 4.2 Stock List Table

Database Desktop

File Edit View Table Record Tools Window Help

Table : C:\Company\customer.db

customer	CustomerId	Name	Surname	Fax	Phone	Email	Address
1	1	HUSEYIN	AR	2301258	2301286	huseyinar@hotmail.com	KURTULUS STREET NO:48
2	2	SALIH	BICER	5545455	3356466	salih@hotmail.com	ALTAYLAR GIRESUN
3	3	TASKIN	KIZIL	3202515	3214564	tkizil@msn.com	ETLIK ANKARA

Record 1 of 3

Fig. 4.3 Customer Table

Database Desktop - [Table : C:\Company\Personel.db]

File Edit View Table Record Tools Window Help

Personel	PNo	Fname	LName	Duty	Department	Salary	Phone	Email	HireDate	Address
1	1	HUSEYIN	AR	MANAGER	TECHNIC SER'	750,00	2301286	huseyin@hotmail.c	14.12.2005	Eskisehir
2	3	KAAN	TASKIN	CONTROL	STOCK	2.000,00	2123245	kaan@yahoo.com	12.12.2005	Istiklal Street Ankara
3	4	HARUN	BIRLIK	ACCOUNT	OFFICE	1.000,00	3234564	harun@yahoo.com	01.01.2006	BALIKLIGOL URFA

Record 1 of 3

Fig. 4.4 Personnel Table

Database Desktop - [Table : C:\Company\Company.db]

File Edit View Table Record Tools Window Help

Company	CompanyId	Name	Phone	Fax	Web	Adress
1	1	ARENA	0212-3254542	0212-3213251	www.arena.com.tr	YENI STREET NO:51 KADIKOY ISTANBUL
2	3	GOLD	0212-2352154	0212-2314512	www.goldpc.com	TAKSIM NO:32 ISTANBUL
3	4	CASPER	0212-2313542	0212-2105255	www.casper.com.tr	AKSARAY NO:12 ISTANBUL
4	5	DELTA	0212-2125452	0212-2154121	www.delta.com.tr	LEVENT

Record 1 of 4

Fig. 4.5 Company Table

Database Desktop - [Table : C:\Company\sale.db]

File Edit View Table Record Tools Window Help

sale	SaleId	CustomerID	ProductId	Date	Price	Quantity
1	1	1,00	4,00	17.12.2005	118,00	2,00
2	2	1,00	2,00	08.12.2005	106,20	1,00
3	3	1,00	4,00	04.01.2006	0,00	0,00
4	4	2,00	8,00	04.01.2006	526,28	1,00
5	5	1,00	1,00	07.01.2006	217,12	2,00
6	6	1,00	1,00	07.01.2006	217,12	2,00

Record 1 of 26

Fig. 4.6 Sale Table

Database Desktop - [Table : C:\Company\Order.DB]

File Edit View Table Record Tools Window Help

Order	OrderId	OName	Properties	Quantity	Company	Price	ODate	ADate
1	1	HP DESKJET 3845	WELL SPEED AND QUALITY	15,00	ARENA	124,00	08.05.2005	15.05.2005
2	2	KINGSTON 512MB 400MHZ	HIGH SPEED	20,00	GOLD	35,00	10.06.2005	25.06.2005
3	3	80GB HDD 7200RPM SATA	SERIAL ATA HAS GOOD SPEED	5,00	GOLD	59,30	05.12.2005	05.01.2006

Record 1 of 3

Fig. 4.7 Order Table